

REALISATION RANKING AND WORD ORDERING IN THE CONTEXT OF LESSER RESOURCED LANGUAGES

Yasaman Motazedi

BSc Software Engineering, Azad University, Tehran

MSc Artificial Intelligence, Azad University, Tehran

This dissertation is presented for the degree of

Doctor of Philosophy

at

Department of Computing

Macquarie University



MACQUARIE
University
SYDNEY • AUSTRALIA



**CENTRE FOR
LANGUAGE
TECHNOLOGY**

December 2017

Contents

1	Introduction	1
1.1	(Re)ranking the Output of the Surface Realiser	5
1.2	Generation from Bag-of-Words	10
1.3	Contribution	12
1.4	Thesis Outline	13
2	Literature Review	15
2.1	Overview of Natural Language Generation (NLG) Systems	15
2.1.1	NLG systems	17
2.2	Data-Driven Methods: Surface Realisation Ranking	18
2.2.1	Realisation Ranking using Log-linear Models	20
2.2.1.1	Realisation Ranking using Log-linear with Head-driven Phrase Structure Grammar (HPSG)	21
2.2.1.2	Realisation Ranking using Log-linear with Lexical Functional Grammar (LFG)	24
2.2.1.3	Realisation Ranking using Log-linear with Combinatory Categorical Grammar (CCG)	34
2.3	Data-Driven Methods: Surface Realisation using Other Input	35
2.3.1	Data-driven Dependency Parsing using Spanning Trees	38
2.3.2	Parse Trees as Intermediate Structures for String Regeneration	40
2.4	Creative NLG Systems	45
2.5	Summary	50
3	Grammatical Structures for Reranking	51
3.1	Introduction	51
3.2	Related Work	52
3.2.1	Parser-based Ranking	52
3.2.2	Unsupervised Parsers	53
3.3	Experimental Setup	56
3.3.1	Data and Evaluation	56
3.3.2	Parsers and Language Models	57
3.3.3	Models	58

3.3.3.1	Base Models, Supervised and Unsupervised . . .	58
3.3.3.2	Unsupervised with Reliability-based Selection . .	61
3.3.3.3	Unsupervised with IG Selection	62
3.4	Results	65
3.5	Summary	73
4	Combinations with Internal Structures for Reranking	77
4.1	Introduction	77
4.2	Features from Intermediate Structures	78
4.2.1	C-structure Features	78
4.2.2	F-structure Features	79
4.2.3	Other Considerations	86
4.2.3.1	Grammar Size	86
4.2.3.2	Incorporation of Features from Statistical Parsers	88
4.2.4	Models	88
4.3	Experiment Configuration and Experiment Results	91
4.3.1	Data Preparation and Experiment Setup	91
4.3.2	Experiment Results	92
4.3.2.1	Base Models	92
4.3.2.2	Combination of C- and F-structure Models . . .	95
4.3.3	Discussion on the Results of the Small Grammar	99
4.3.4	Summary	102
5	An ILP Framework for String Regeneration	105
5.1	Introduction	105
5.2	Related work	109
5.2.1	Brief Review of Integer Linear Programming (ILP)	109
5.2.2	ILP in Graph-based Parsing	113
5.3	An ILP Framework for Surface Realisation	115
5.4	Defining an Objective Function	118
5.5	Enforcing Tree Structure	118
5.6	Imposing Linguistic Constraints	121
5.6.1	A Hard Encoding	121
5.6.1.1	Dependency Tree Constraints	121
5.6.1.2	Branching Factor Constraints	122
5.6.2	A Flexible Encoding	127
5.6.2.1	Multi-objective Optimisation	127
5.6.2.2	Reformulation of Minimum Spanning Tree (MST) to a Multi-objective Problem	132
5.7	Conclusion	134
6	Applying the ILP Framework	137
6.1	Introduction	137

6.2	Related Work: Online Large Margin Learning	138
6.3	Feature Selection and Feature Extraction	143
6.4	Training	147
6.5	A Small-Scale Trial and Illustration	153
6.6	Experiment Setup	153
6.6.1	General Details	153
6.6.2	Models	157
6.6.3	Evaluation Metrics	158
6.7	Evaluation Results	161
6.7.1	ILP	161
6.7.2	QP	165
6.7.3	QP-MIRA	168
6.8	Comparison and Conclusion	172
7	Conclusion	177
7.1	Summary of Findings	177
7.2	Limitations and Future Outlook	179
A	Small-Scale Trial	181

List of Tables

2.1	The distribution of content and structure tasks per module in ‘consensus architecture’, reproduced from Figure 3.1 from Reiter and Dale (2000)	16
2.2	Features used by Riezler et al. (2002), Riezler and Vasserman (2004) for the disambiguation of English ParGram LFG parses (Cahill et al., 2007a)	28
2.3	Features templates used by Cahill et al. (2007b) for semi-automatic feature construction for parse disambiguation (Cahill et al., 2007b)	32
2.4	SR-Task teams and systems. ^x = resubmitted after fixing software bugs; ^y = late submission (Belz et al., 2011)	36
2.5	Comparison between Deep and Shallow representation. * = where necessary	37
3.1	The generative process for model parameters and parses, where s is an observed coarse symbol, z is a hidden refined subsymbol, and x is an observed word. s' and z' represent the parent of the current node symbol and subsymbol. In the above GEM, DP, Dir and Mult refer respectively to the stick breaking distribution, Dirichlet process, Dirichlet distribution, and multinomial distribution. Reproduced from Table 2 in Naseem et al. (2010).	54
3.2	The manually-specified universal dependency rules used by Naseem et al. (2010)	55
3.3	Classification scores for supervised parse features and large LM on random : accuracy on parse features; parse features plus large l-lm4 LM over these sentences; parse features plus large l-lm2 LM over these sentences	66
3.4	Classification scores for supervised parse features and large LM on greatest : accuracy on parse features; parse features plus large l-lm4 LM over these sentences; parse features plus large l-lm2 LM over these sentences	66

3.5	Classification scores for unsupervised parse features and large LM on <i>random</i> : accuracy on parse features; number of sentences where feature vectors differ for positive and negative examples; accuracy over effective test set.	66
3.6	Classification scores for unsupervised parse features and large LM on <i>random</i> : number of sentences where feature vectors differ for positive and negative examples; accuracy of language models (individually, or combined with unsupervised model) over effective test set.	67
3.7	Classification scores for unsupervised parse features and small LM on <i>random</i> : number of sentences where feature vectors differ for positive and negative examples; accuracy of language models (individually, or combined with unsupervised model) over effective test set.	67
3.8	Classification scores for unsupervised parse features selected by reliability on <i>random</i> : threshold cut-off accuracy on parse features; number of sentences where feature vectors differ for positive and negative examples (effective test set)	69
3.9	Highest- (left) and lowest- (right) ranking dependency features based on reliability measure. For each, columns represent the dependency; the number of times it was correct in an unsupervised parse; and the number of times it occurred in the gold standard.	69
3.10	Classification scores for unsupervised parse features selected by IG: threshold cut-off; accuracy on parse features; number of sentences where feature vectors differ for positive and negative examples (effective test set); large LM score over these sentences	70
3.11	Highest ranking features based on IG.	71
4.1	Reranking templates extracted from functional structure (f-structure) features	85
4.2	List of model names; + and - represent the presence or the absence of the given feature.	88
4.3	List of model names; + and - represent the presence or the absence of the given feature.	88
4.4	Naming convention for various models using dependency structure	91
4.5	Results of reranking with c-structure features extracted from the large grammar (a), and the small grammar (b).	93
4.6	Results of reranking with f-structure features extracted from the large grammar (a), and the small grammar (b).	94
4.7	Naming convention for various models using dependency structure	96
4.8	Reranking results for the combination of the best c-structure and f-structure models	96

4.9	Results of reranking using various combinations of (a) supervised and (b) unsupervised dependency parser features added to CS-structure features extracted from the small grammar	97
4.10	Results of reranking using various combinations of (a) supervised and (b) unsupervised dependency parser features added to CL-structure features extracted from the large grammar	98
4.11	Results of reranking using various combinations of (a) supervised and (b) unsupervised dependency parser features added to c-/f-structure features extracted from the small grammar	99
4.12	Results of reranking using various combinations of (a) supervised and (b) unsupervised dependency parser features added to c-/f-structure features extracted from the large grammar	100
4.13	Summary of ranking results for various combinations of features from external resources with (a) the small grammar and (b) the large grammar. Regardless of the size of the grammar, and the best of the external features didn't improve over just internal.	100
4.14	Encoded PARENT & CHILD (CS-PC) and GRANDPARENT, PARENT & CHILD (CS-GPC) features along with their count. F_CS_3–F_CS_14 and F_GP_3–F_GP_14 are following are following FRAGMENTS- w_i and FRAGMENTS-FRAGMENTS- w_i templates, where w_i can be replaced by any word from sentence (2.1). As discussed above, the small grammar is unable to assign a detailed structure including grouping words into phrases and so on. Consequently, the features in such a shallow tree becomes quite repetitive, e.g., FRAGMENTS-TOKEN has been observed 13 times	101
5.1	Mode and maximum values of the branching factor for WAKE extracted from google syntactic n-gram.	124
5.2	(a) Valency templates for the word <i>eat</i> . (b) Probability of right and left valencies are calculated.	133
5.3	Weighted mean (\bar{v}) and weighted standard deviation (s) of the left and the right valencies of the head words in sentence (5.4)	133
6.1	Features used in dependency parser (McDonald, 2006) replaced by p and c prefixes indicate parent node and child node respectively. wrd is the actual word and pos is the Part of Speech (PoS) tag of the given node. c-pos+1 represents the PoS tag to the right of the child node while c-pos-1 refers to the PoS tag to the left of the child node. The example column is illustrating such features based on the dependency tree in Figure 6.1.	140

6.2	Features used in the QP-MIRA model. p- and c -prefixes indicate parent node and child node, respectively. word is the actual word and pos is the PoS tag of the given node. c-pos+1 represents the PoS tag to the right of the child node while c-pos-1 refers to the PoS tag to the left of the child node. The example column used the dependencies in Figure 6.1.	145
6.3	Extracted features for the dependency (root,eats) from the complete graph presented in Figure 5.2 using FR-WO scheme. Notation: pw: parent word, cw: child word, pp: parent PoS tag, cp: child PoS tag.	150
6.4	Weight matrix for sentence (6.1) using F-WO model.	155
6.5	Weight matrix for sentence (6.1) using R-WO model.	156
6.6	List of model names; presence of each constraint is indicated by 1	157
6.7	Comparison of the BLEU score for the six hypothetical systems	160
6.8	The ILP evaluation result: coverage and BLEU% over T_s and T along with the corresponding CLE and AB scores. The back-off column is over the whole set T and represents the ILP system backing off to CLE or AB in case of failure.	163
6.9	Coverage and BLUE score of QP over T_s as well as the over all BLEU score along with the baseline algorithms' scores over the corresponding test sets. The back-off column is over the whole set T and represents the QP system backing off to CLE or AB in case of failure.	166
6.10	QP_MIRA system is trained and tested over MIRA1–MIRA7 models which are based on three ILP models M1, M15 and M16 (discussed in the previous section) and differ in the number of iterations used for training MIRA (learning weights) and the threshold used for controlling the incremental generation of cycle-forbidding constraints.	169
6.11	Coverage and BLUE score of QP over T_s as well as over T , along with the baseline algorithms score over the corresponding test sets.	169
6.12	Comparison of our baselines (CLE and AB) with the best of ILP, QP and QP-MIRA over the whole test set, T , where +CLE and +AB specify the back-off algorithms.	175
A.1	Objective function (weight matrix) for sentence (6.1) in the first iteration of training with FR-WO features.	181
A.2	Objective function (weight matrix) for sentence (6.1) in the second iteration of training with F-WO features.	183
A.3	Objective function (weight matrix) for sentence (6.1) in the third iteration of training with FR-WO features.	184
A.4	Objective function (weight matrix) for sentence (6.1) in the fourth iteration of training with FR-WO features.	185

A.5	Objective function (weight matrix) for sentence (6.1) in the first iteration of training with F-WO features.	186
A.6	Objective function (weight matrix) for sentence (6.1) in the second iteration of training with F-WO features.	187
A.7	Objective function (weight matrix) for sentence 6.1 in the third iteration of training with F-WO features.	188
A.8	Objective function (weight matrix) for sentence (6.1) in the fourth iteration of training with F-WO features.	189

List of Figures

1.1	Example forecasts from FOG.(Reiter and Dale, 2000)	1
1.2	Structured numerical data used for generating weather forecasts, reproduced from Figure 1.2 in Reiter and Dale (2000)	2
1.3	Top three parse analyses for Sentence (1.1)	7
1.4	XLE representation for Sentence (1.1).	8
1.5	XLE internal representation for Sentence 1.7.	9
2.1	An NLG system architecture. (Reiter and Dale, 2000)	16
2.2	A sets of a generated sentences using LinGO ERG. (Velldal and Oepen, 2005)	19
2.3	Two steps of NLG process for NLG1, NLG2 and NLG3 with corresponding input and output. Words starting with \$ are attributes. (Ratnaparkhi, 2000)	21
2.4	Sample dependency tree for the phrase <i>evening flights from Chicago in the afternoon</i> + and - signs indicate right and left child, respectively.(Ratnaparkhi, 2000)	21
2.5	HPSG feature structure for pronoun <i>she</i>	22
2.6	HPSG feature structure for pronoun <i>walks</i>	22
2.7	constituent structure (c-structure) for sentence (2.1)	25
2.8	f-structure for sentences (2.1)	26
2.9	f-structure for sentence (2.2)	30
2.10	Alternative realisations for the f-structure in Figure 2.9 generated by Xerox Linguistics Environment (XLE).	31
2.11	Agreement as a ranking problem.(Rajkumar and White, 2010) . .	34
2.12	A step by step example of the Chu-Liu-Edmonds (CLE) algorithm to get the MST for Sentence (5.1). (a) create initial digraph and assign weights to each dependency; (b) greedily choose incoming dependencies with minimum weight; (c) contract the cycle into a temporary node <i>t</i> and recalculate the weights; (d) once again, greedily choose incoming dependencies with minimum weight which results in a valid tree; (e) revert <i>t</i> to its original building nodes to get the MST (McDonald et al., 2005b).	39

2.13	A complete digraph for a bag-of-words of size n . Each N_i represents a word and each w_{ij} denotes the weight for a dependency which N_i is the head and N_j is the dependent	41
2.14	The spanning tree of the sentence: <i>People still remain displaced in Dili, burdening hosts</i> . Pages 188–189 of Wan et al. (2009) describe the process step-by-step.	43
2.15	Comparison of CLE and Hungarian algorithms output. Comparing trees in 2.15(a) and 2.15(b) shows the effectiveness of using branching factor in the later.	44
2.16	feed-forward neuralnetwork with two hidden layers, reproduced from Figure 2 from Goldberg (2015).	46
2.17	2.17(a) the unrolled graphical representation of an RNN, reproduced from Figure 5 from Goldberg (2015). 2.17(b) The rolled graphical representation of an RN, reproduced from Figure 6 from Goldberg (2015).	47
2.18	Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: (1) Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). (2) Sequence output (e.g. image captioning takes an image and outputs a sentence of words). (3) Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). (4) Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). (5) Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case there are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like (reproduced from an example provided in http://karpathy.github.io/2015/05/21/rnn-effectiveness/)	49
3.1	A tree showing head information (Charniak, 2001).	52
3.2	Unsupervised dependency tree for a sample sentence. Edges in bold are the dependencies that are identified by both supervised and unsupervised parser.	59
3.3	Supervised dependency tree for sample sentence of Fig 3.2. Edges in bold are the dependencies that are identified by both supervised and unsupervised parser.	60
3.4	Unsupervised dependency tree for another sample sentence. Edges in bold are the dependencies that are identified by both supervised and unsupervised parser.	63

3.5	Supervised dependency tree for sample sentence of Fig 3.4. Edges in bold are the dependencies that are identified by both supervised and unsupervised parser.	64
3.6	Supervised dependencies (lower arcs) vs unsupervised dependencies (upper arcs) for sentence 3.2	72
4.1	Examples of the realisation reranking c-structure features; subtrees are extracted from the c-structure presented in Figure 2.7: (a) CS-PC, (b) CS-GPC, (c) SIBLINGS (CS-SB), and (d) PARENT & CHILDREN (CS-NC).	79
4.2	ATTRIBUTES (FS-ATTRS)	80
4.3	ATTRIBUTE-VALUE PAIRS (FS-ATTR-VAL)	81
4.4	ADJACENT ATTRIBUTES (FS-ADJ-ATTR)	82
4.5	SUB-ATTRIBUTES (FS-SUB-ATTR)	82
4.6	AUNT ATTRIBUTES (FS-AUNT-ATTR)	83
4.7	LEXICAL SUB-CATEGORIES (FS-LEX-SUBCAT)	84
4.8	VERB ARGUMENTS (FS-VERB-ARGS)	84
4.9	c-structure for sentence (2.1) using ParGram Starter Grammar	87
4.10	f-structure for sentence (2.1) using ParGram Starter Grammar	87
4.11	(a) is the dependency tree for sentence (2.1). In (b), (Vinken, died) pair is an example of lexicalised dep feature. The (VBD, NNP, NN) triple in (c) represents an unlexicalised instance of the ancestor template. (d) The lexicalised pair (former, Elsevier) are two siblings that share chairman as their mother. Finally, (e) represents the mother-siblings template by the (chairman, former, Elsevier) triple.	90
5.1	The bag-of-words (5.2) annotated with gold standard dependencies that can be served as an input to the ordering task.	106
5.2	Comparison of the two MSTs generated by Wan et al. CLE-based approach and ILP for Sentence (5.3). Note that base NPs are represented by their head. e.g. <i>cat</i> is the head of <i>the cat</i> base NP.	107
5.3	Graphical solution of the problem in Example (5.2), reproduced from Bertsimas and Tsitsiklis (1997).	111
5.4	All the possible dependencies for two words. (a) and (b) describe Forward word order. In (a) w_1 precedes w_2 , and the dependency is right pointing r , i.e, the right-most word w_2 is the dependent. Word order in (b) is identical to (a) but the dependency direction is left pointing l which makes w_1 the dependent and w_2 the head. In (c) w_2 precedes w_1 and the dependency is right pointing. (d) has the similar word order to (c) but the dependency is left pointing.	116
5.5	A complete weighted digraph for two words; it specifies word-order and dependency direction at the same time.	117

5.6	A complete weighted digraph for three words with word order and dependency direction specified at the same time	120
5.7	An example of a dependency tree where x_i has violated its maximum branching factor and caused some of the other nodes not to have the expected number of dependents.	123
6.1	Dependency tree for sentence <i>John hit the ball with the bat.</i>	139
6.2	Gold standard dependency tree.	142
6.3	First best predicted parse tree.	142
6.4	Second best predicted parse tree.	143
6.5	Some of the encoded features for 4 possible dependencies given two words A and B. These features are defined in Table 6.2 and extracted from a model trained over the F-WO plane of training data. We colour-coded the features for easier reference. All the purple features — shared between d_2 and d_3 — like 66, must be of basic Uni-gram and/or basic Bi-gram features, as these two dependencies share the same dependency direction: A is the head and B is the dependent; similar argument is valid for blue features for d_1 and d_4 . The features highlighted with orange and gray must be of the Extended feature group, since they are unique to the dependency. We did not highlight unique dependencies for d_3 and d_4 for readability purposes. Represented with fewer unique features indicates that the last two dependencies have no exact match in the training set, but still partially matched with some instances, e.g. 18 represents p-pos, c-pos, dir feature (see Figure 6.3 for feature sets prior to encoding).	146
6.6	ILP-MIRA/QP-MIRA diagram	148
6.7	MST_MIRA diagram	154
6.8	ILP generated tree with initial weights. Each Arc's label is the dependency weight associated with it, highlighted in Table 6.4. . .	154
6.9	ILP Each Arc's label is the dependency weight associated with it, highlighted in Table 6.5.	155
6.10	A subset of test set sentences along with their reordered alternative generated by ILP and QP systems. These sentences will be used to show the brevity penalty effect on the total BLEU a subset with missing sentences.	159
6.11	ILP coverage rate broken down by the length bucket: SS: 1–10 words MS: 10–20 words, LS: 20–25 words, VLS: 25–30 words, XLS: 30+ words.	162
6.12	Comparison of the ILP system and the baseline algorithms over T_s . . .	164
6.13	Overall comparison of the ILP system and the baseline algorithms. . .	165

6.14	QP coverage rate broken down to length bucket: SS: 1–10 words MS: 10–20 words, LS: 20–25 words, VLS: 25–30 words, XLS: 30+ words.	166
6.15	Comparison of the QP system and the baseline algorithms over T_s	167
6.16	Comparison of the QP system and the baseline algorithms over T_s	168
6.17	(a) Quality of generated dependency trees trees using QP-MIRA; (b) QP-MIRA Coverage rate broken down to length bucket: SS: 1–10 words MS: 10–20 words, LS: 20–25 words, VLS: 25–30 words, XLS: 30+ words.	170
6.18	QP-MIRA compared to its baseline algorithm (a) over T_s and (b) over the whole test set T	171
6.19	Comparison of ILP, QP and QP-MIRA when tested with three different constraint sets in terms of their (a) BLUE score over T when backed off to the baseline models, and (b) coverage% over T_s . M1 is the model with tree constraints only; M15 has tree constraints plus PUNC-VALENCY, COORD-VALENCY and PREP- VALENCY constraints; M16 has NOUN-VERB-ARITY in addition to all the M15 constraints.	173
6.20	Comparison of the ILP and QP over the test set, T	174
A.1	ILP generated tree with initial weights. Arcs' labels are depen- dencies' weights	182
A.2	ILP generated tree with initial weights. Arcs' labels are depen- dencies' weights	183
A.3	ILP generated tree with initial weights. Arcs' labels are depen- dencies' weights.	184
A.4	ILP generated tree with initial weights. Arcs' labels are depen- dencies' weights.	185
A.5	ILP generated tree with initial weights. Arcs' labels are depen- dencies' weights	186
A.6	ILP generated tree with initial weights. Arcs' labels are depen- dencies' weights	187
A.7	ILP generated tree with initial weights. Arcs' labels are depen- dencies' weights.	188
A.8	ILP generated tree with initial weights. Arcs' labels are depen- dencies' weights	189

Abstract

Natural Language Generation (NLG) is a subfield of Language Technology, which concentrates on generating human-understandable sentences from machine-oriented input such as databases, knowledge bases or logical forms. This involves making many decisions on the surface representation of sentences, including lexical selection, use of referring expressions, and word order relying on manually constructed grammars.

In this thesis we are focusing on two subtasks of NLG: realisation ranking and word ordering. In this we are motivated by situations where there are few computational resources, and where we need to identify alternative resources and algorithms that guide the text generation task. We investigate two scenarios: one where there is a manual grammar that is involved in the language generation component, and one where there may not be.

With respect to the first scenario, which often contain a component that reranks the output of the grammar based on system-internal representations, we look at finding alternative resources to extract ranking features. As the first step, we look into several supervised statistical parsers to see to what extent trees other than those from the NLG grammar are useful. The next step is moving to unsupervised parsing, a statistical method to generate a parse tree for any given sentence without involving any manual grammar or treebank training. Our findings show that features from supervised parsers and also selected features from unsupervised parses can improve generation ranking.

The second aspect is taking text generator internal structures as the ranking feature and study how effective they are in ranking with respect to the grammar size. Finally, recognising whether features from external resources such as unsupervised parser can be coupled with text generator internal structure to rank generated text. Our experiments reflect that using internal features lead to better ranking than the baseline.

The second scenario is performing partial-tree linearisation in the absence of manually crafted grammars. We introduce a novel Integer Linear Programming (ILP) framework on top of an existing graph-based lineariser as a declarative way of introducing linguistically motivated features into the generation process. This framework accommodates various constraints independently and incrementally. As part of this, we present a novel application of machine learning methods for adjusting ILP parameters in order to improve the quality of generated strings. Comparing with the baseline systems in terms of the coverage and the quality of the generated text, our results prove the superiority of the ILP-based realiser when it applies linguistically motivated constraints over the learnt ILP parameters.

Declaration

The research presented in this thesis is the original work of the author except where otherwise indicated. Some parts of the thesis include revised versions of published papers. This work has not been submitted for a degree or any other qualification to any other university or institution. All verbatim extracts have been distinguished by quotations, and all sources of information have been specifically acknowledged. Chapter 3 of this thesis is mainly based on the revised version of the following published paper:

Yasaman Motazedi, Mark Dras, François Lareau. Is Bad Structure Better Than No Structure?: Unsupervised Parsing for Realisation Ranking. *In Proceedings of the 24th International Conference on Computational Linguistics (Coling'12)*, pages 1811-1830, Mumbai, India, December 2012.

Signed: Yasaman Motazedi

Date: 11 Dec 2017

To Sohrab Khaleghi Rad, for reassuring me to take this major step and standing by me with your big smile all the way through it.

Acknowledgements

First of all, I appreciate my supervisor Mark Dras for the invaluable supervision. Mark, I am truly grateful for your full support that makes this thesis happen in the end. Probably, I never said it out-loud but I am highly indebted to you for expanding my worldview during the course of our many discussions during the past few years.

Thank you to my examiners Dr. Aoife Cahill, Dr. Ben Hachey, and Dr. Karin Verspoor for their pain-staking review of this thesis and their many suggestions for making it a better and more thorough piece of work.

Stephen Wan, thank you for passing me your PhD material to be used as the baseline for the work I did in Chapters 6 and 7 of this thesis. Jette Viethen, thank you for sharing your invaluable advices for thesis writing, specially *Make sure you write one paragraph a day!*, reviewing bits and pieces of my drafts topped up with accommodating me in your sun-lit office.

Not forgetting to mention that I received much intellectual and emotional support from both the current and past members of the Centre of Language Technology at Macquarie University. I extend my thanks to, not in a particular order, Teresa Lynn, Jojo Wong, Jette Viethen, Mary Gardiner, Susan Howlett, Marta Vila Rigat, Benjamin Börschinger, Simon Zwartz, François Lareau, Diego Mollá, Rolf Schwitter, Robert Dale, Matthew Honnibal, Bevan Jones, Shervin Malmasi, Andrew Lampert, Stephen Wan, Abeed Sarker, Chris Rauchle, Steve Cassidy, Marc Tilbrook, Mehdi Parviz, Jason Naradowsky, Tony Zhao, Lan Du, Mac Kim. I appreciated all the great moments that we shared together in lunch-breaks reading groups, meetings, and conferences.

There is no way that I can forget the impact that Mehrnoush Shamsfard my MSc supervisor had on me to pursuit my passion for Language Technology. So, I would like to take this opportunity to express my gratitude to you once again.

Many thanks to my friends Sahar Mohseni, Maryam Azadi Gonbad, Hakimeh Fadaei, and Chakaveh Saedi. Your long-distance support and continuous follow-ups had a double effect of motivating me to finish and also filling me with the sense of appreciation for our past collaboration on various research projects at different

points in time.

Thank you Han Xu for your friendship and mutual understanding of the critical task of splitting the effort and devotion between developing the NLG engine for OwnersAdvisory™, Macquarie Bank and writing up a PhD thesis. I have almost missed our daily thesis-scrolling ritual to share the amount of work had been carried out after work the previous evening.

Many thanks to my Sydney-based friends, Farnaz Ghadrddan whom persistently invited me to various gatherings knowing the highly likeliness of hearing “Sorry guys :(it’s another weekend in uni for me. Have fun!” For the last couple of years, you did a great job in keeping me in the circle.

I appreciate my parents, Fariba Mahdavi and Abdolai Motazedi for bringing me up this ambitious and driven. I thank my sister Niloofar Motazedi, for keeping the sister-talk routine with the additional PhD progress report load, regardless of the time-zone difference. My special thanks to my in-laws Fereshteh Ghazanfari and Ahmad Khaleghi Rad for your continuous encouragement, your exemplar understanding of life of a PhD student and scheduling every one of your visits to Sydney with respect to my various PhD deadlines.

It is almost impossible to express my gratefulness towards my husband Sohrab Khaleghi rad. Sohrab, your unconditional love, selfless support in all possible ways and impeccable sense of humour made this journey, way more manageable and even a memorable one. Everyone can tell how severely you have been traumatised, just by proof-reading this thesis:

“Wow! reading thesis is hell of job — very boring! Writing a thesis should be mother of god, hell of seven hells job. I really feel for you right now!”

Chapter 1

Introduction

Natural Language Generation (NLG) as per definition of Reiter and Dale (2000) is “a subfield of Artificial Intelligence (AI) and computational linguistics that focuses on computer systems that can produce understandable texts in English or other human languages. Typically starting from some nonlinguistic representation of information as input, NLG systems use knowledge about language and the application domain to automatically produce documents, reports, explanations, help messages, and other kinds of texts.” The weather report presented in Figure 1.1 is a sample of an automatically generated text by FOG (Goldberg et al., 1994) from structured numerical data similar to the dataset depicted in Figure 1.2.

EAST BREVOORT
EAST DAVIS
GALE WARNING CONTINUED.
WINDS SOUTH 30 TO GALES 35 DIMINISHING TO SOUTH WINDS 15
EARLY FRIDAY MORNING. WINDS DIMINISHING TO LIGHT FRIDAY
EVENING.RAIN TAPERING TO SHOWERS THIS EVENING AND CON-
TINUING FRIDAY. FOG DISSIPATING THIS EVENING.
OUTLOOK FOR SATURDAY... LIGHT WINDS..

Figure 1.1: Example forecasts from FOG.(Reiter and Dale, 2000)

There have been many different architectures proposed for NLG systems (De Smedt et al., 1996), but the most widely agreed-upon one is the pipeline architecture, also described by Reiter and Dale (2000). It is frequently referred to as the ‘consensus architecture’ which consists of the following three components:

1. Document planner: determines the content and defines document structure based on possible data availability in the input, i.e., what to say.
2. Microplaner: defines semantic structure, marks possible referring expressions and determines sentence aggregations, i.e., how to structure it.

96,122,1,5,2.00,200,-14.41,-3.668,-1.431,.345,1023,15.41,15.82,20.07,-11.1,-2.878,104.2,.28,153.6,53.19,0,16.26
 96,122,1,5,2.25,215,-10.72,-3.241,-1.35,.152,1023,15.3,15.78,20.07,-11.42,-2.762,105,.208,98.2,.822,0,17.05
 96,122,1,5,2.50,230,-8.37,-1.282,-.904,2.15,1022,15.3,15.71,20.05,-11.66,-3.206,104.4,.2,141.6,42.96,0,17.7
 96,122,1,5,2.75,245,-12.81,-2.11,-1.067,2.119,1022,15.33,15.79,19.99,-11.15,-3.093,104.8,.2,186.5,11.32,0,17.81
 96,122,1,5,3.00,300,-13.68,-3,-1.35,1.075,1022,15.36,15.79,19.96,-10.63,-3.005,104.6,.402,285.8,61.45,0,18.47
 96,122,1,5,3.25,315,-10.2,-2.457,-1.13,-.73,1022,15.32,15.66,19.92,-11.17,-3.263,103.6,.304,354.7,36.29,0,19.03
 96,122,1,5,3.50,330,-9.33,-1.353,-.942,.902,1022,15.21,15.62,19.9,-10.95,-2.903,104.3,.313,302.2,34.69,0,19.16
 96,122,1,5,3.75,345,-7.29,-.285,-.76,2.048,1022,15.24,15.63,19.87,-10.68,-3.27,104,.252,313,29.7,0,19.61
 96,122,1,5,4.00,400,-6.822,-.365,-.653,1.531,1022,15.25,15.63,19.83,-9.93,-3.316,104,.331,274.2,52.98,0,20.42
 96,122,1,5,4.25,415,-8.78,-.65,-.747,1.602,1023,15.35,15.66,19.79,-9.77,-2.656,103.3,.253,247.7,10.99,0,21.08
 96,122,1,5,4.50,430,-8.73,-.641,-.741,1.785,1023,15.46,15.81,19.75,-9.16,-2.782,103.7,.2,295.29,15,0,21.3
 96,122,1,5,4.75,445,-11.45,-2.671,-1.03,-.456,1022,15.46,15.82,19.74,-8.81,-2.464,103.7,.2,355.3,23.98,0,21.65
 96,122,1,5,5.00,500,-13.12,-4.3,-1.306,-1.359,1022,15.42,15.75,19.76,-9.39,-2.49,103.4,.2,20.67,.188,0,21.83
 96,122,1,5,5.25,515,-13.62,-4.621,-1.344,-.842,1022,15.32,15.67,19.81,-9.47,-2.703,103.7,.2,20.65,.183,0,21.98
 96,122,1,5,5.50,530,-13.8,-3.534,-1.325,.943,1022,15.23,15.61,19.86,-10.92,-3.384,103.9,.2,20.65,.183,0,22.14
 96,122,1,5,5.75,545,-14.7,-3.748,-1.419,.385,1022,15.06,15.47,19.9,-11.62,-2.868,104.4,.2,341.6,18.6,0,22.36
 96,122,1,5,6.00,600,-13.61,-2.315,-1.287,2.038,1022,14.98,15.42,19.9,-12.37,-3.092,104.7,.2,298.6,5.173,0,22.54
 96,122,1,5,6.25,615,-14,-2.894,-1.293,.669,1022,14.92,15.36,19.88,-12.48,-3.808,104.7,.591,320.3,21.07,0,22.87

The 22 data values in each record are as follows: the year; the day of the year as a value between 1 and 365; the month of the year; the day of the month; the time in 24 hour notation; the time in hours and minutes; four radiation readings; the barometric pressure; the temperature, referred to as 'dry bulb temperature'; the wet bulb temperature, which is used to calculate humidity; the soil temperature; the soil heat flux at two depths; relative humidity; average wind speed; wind direction; the standard deviation of wind direction; precipitation within the 15 minute period; and amount of sunlight.

Figure 1.2: Structured numerical data used for generating weather forecasts, reproduced from Figure 1.2 in Reiter and Dale (2000)

3. Surface realiser: converts the semantic structure into human readable text with correct word forms and performs subject elision or sentence aggregation where possible, i.e., how to lexicalise it.

This architecture conceptually distinguishes between these three components since each of them contributes to the quality of the generated text at a different level. Not all computational linguistics resources and methods required by each component have been explored equally. The document planner requires algorithms to choose interesting aspects of data and decide how much information must be communicated. These algorithms are numeric, e.g. spotting a significant change in a trend, and highly relying on historical data. Structuring the text can also be achieved through analysing similar reports where enough text is available e.g. meteorological forecasts; otherwise this task becomes a human expert job. This implies that automating this component as a general purpose tools seems to be very challenging in light of the lack of historical data/reports. The second component can be regarded as a function that maps each piece of data into a semantic structure and so relies on knowledge bases and language-specific resources which assist with semantic structure selection. The surface realiser is the most grammar-aware component amongst all the three and particularly affects the readability of the generated text. Realisers have been popular amongst researchers since they can be found in other Natural Language Processing (NLP) applications such as abstractive summarisers, paraphrase generation and machine translation systems. Realisers have traditionally relied on grammars which are mainly hand crafted — very expensive to build. There have been recent efforts to improve the output of realisers by attacking the problem from various perspectives, for instance: improving the quality of the grammar, introducing reranking algorithms where there is more than one sentence per piece of data, or combining traditional generation methods with statistical methods to generate the sentences.

There are also efforts that follow other architectures, although these in some cases are in the very early stages. The quotation below is a piece of automatically generated text to experiment with the ability of computers to generate creative articles using a Recurrent Neural Network (RNN) trained over a corpus of Adrienne Lafrance's articles.

“She told me and more like modernings in our computer. Of course, this is not a human work. That's the web that's selfies that would be the moon is that they'll make it online. That's not only more importantly changed and most of them in all of those and questions about their factors. For example, as far several cases, all this kind of regulations for information? that's the person who painted itself that's the technological change of human process... ”(Lafrance, 2016)

Not only the is the text not quite grammatical but it also does not follow any specific topic and has no coherent semantics behind it. This implies that resources that a

computer requires for being a creative writer have not been developed yet. There have been recent efforts to close this gap by exploiting the synergy between NLG and the Semantic Web (SW), like the Web-NLG project¹ and the content selection challenge from open semantic web data², to develop robust and high quality NLG systems capable of producing natural sounding text from SW data (Bouayad-Agha et al., 2013). In this thesis, however, we focus on the well-developed, traditional architecture, and specifically on the realisation component, with our aim being to incorporate improved data-driven approaches. Our motivations will be discussed shortly after an overview of approaches to NLG systems that are in line with the traditional consensus architecture.

Template-based approach One of the early approaches to automatic text generation is template-based approaches. In this approach, a set of templates must be prepared in advance. Then, at the generation time, a specific template is picked and the content from the knowledge-base or data source would fill in the gaps. YAG (Yet Another Generator) (Channarukul, 1999), for instance, extended the traditional template-based approach by allowing templates to embed several types of control expressions, in addition to simple string values. As per definition, this approach suffers from inflexibility due to being bound to the predefined templates.

Rule-based approach Another traditional approach to address this problem is rule-based techniques which can be dated back to Kafka (Mauldin, 1984) which was a rule-based single sentence generation system. Systems developed with this approach are complicated and very expensive to maintain. However, being deterministic, such systems are still popular for commercial purposes like financial and legal documents. Recently, Bauer et al. (2015) introduced a multilingual text generator system that uses a set of rules to perform content selection from structured data. The selected content is then represented by a semantic model which is later converted to target-language text by applying language-specific rules. Both of the discussed approaches suffer from inflexibility and demand a high level of human involvement in development and maintenance phases. These two pain points have directed research towards more intelligent/machine-based approaches.

Symbolic grammar approach Deploying manually-crafted grammars to find the best ordering for a given bag-of-words is the first of these approaches. For instance ParGram (Butt et al., 2002) grammars are deployed in the XLE framework for both parsing and generation. A key characteristic of hand-crafted grammars in

¹This is a three year research project, started at October 2014, <http://talcl.loria.fr/webnlg/stories/about.html>

²This challenge was a part of Generation Challenges 2013 <http://www.taln.upf.edu/cschallenge2013/>

their basic form is that they are symbolic in nature. Generation will thus typically result in multiple outputs: there are multiple alternatives when deciding for a given non-terminal. Considering this fact, there will be several sentences generated for a given linguistic representation of a sentence. The number of generated sentences has a direct relationship with output sentence length and the quality of the grammar. This implies that such a generator requires a module to assign rank to the generated sentences. The main disadvantage is the need to have a symbolic grammar which is costly to build.

Statistical approach Alternatively, statistical methods have been proposed to bypass the use of a symbolic grammar and computational and linguistic resources for the purpose of text generation. Machine learning methods such as maximum entropy and annotated corpora are the key elements of this approach. A model is trained over a large set of features extracted from a corpus and applying the model to a given bag-of-words would result in the generated text. As Oberlander and Brew (2000) proposed, features should capture either of the following properties: (1) characteristics of a fluent and coherent text, or (2) characteristics of undesirable structures. Zhang (2013) introduced a statistical text generation model where the input varies in a cline from unstructured data, e.g., bag-of-words, to a fully defined structure, e.g., dependency tree of the whole sentence.

Unlike the highly engineered template-based and rule-based systems that produce one surface text per semantic representation, the last two stochastic approaches are less expensive to build and maintain but tend to produce multiple grammatical sentences for a given input. As mentioned above, this produces a need to rank this multiple sentences in order to choose the one to output. These two approaches are the focus of this thesis, and the ones we expand on below.

1.1 (Re)ranking the Output of the Surface Realiser

Ranking refers to the process of assigning priority to multiple candidates based on some measure of goodness; often this measure of goodness is a function of the candidate's distance from a gold standard. One of the most well-known applications for ranking in NLP is parse ranking. It is very common for a sentence to have more than one parse tree, due to for example structural or lexical ambiguity. The ranker orders generated alternatives so that the higher the rank is, the closer the tree is to the gold standard for the given input. For example, the symbolic LOGON online parser³ produces 29 analyses for Sentence (1.1); the top three analyses are shown in Figure 1.3. The highest ranked analysis (Figure 1.3(a)) is clearly distinguishable from the next two alternatives and is due to ambiguity at the

³<http://erg.delph-in.net/logon>

grammar level. However, Figures 1.3(b) and 1.3(c) depict ambiguity in the crafted grammar level where two exact structures only differ in the root label.

(1.1) Try pressing return in this window!

Realisers are often viewed as the converse of parsers: The former is transforming a structure into a sentence, whilst the latter assigns a structure to a sentence. Similarly, realisers must be paired with rankers since they are expected to produce multiple grammatical sentences where some of them are paraphrases but the rest are misleading. The number of generated candidates has a direct relationship with the complexity of the input and the system's grammar rule set. To illustrate this, Xerox Linguistics Environment (XLE) and LOGON online generator are deployed to regenerate Sentence (1.1); in this case the input will be the syntax tree that the parser has chosen as the best.

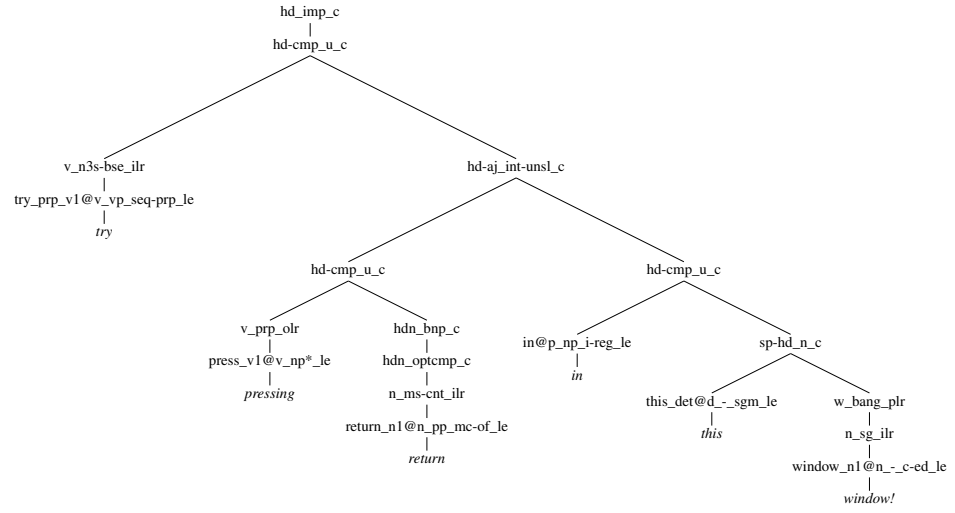
Figure 1.4 is the former system's internal representation of the sentence that will be linearised to a sentence. The produced sentence (Sentence (1.2)) is almost identical to the original sentence except the end of sentence exclamation mark has been replaced by period. The XLE framework and its internal representations, as the basis for one component of the work in this thesis, will be discussed in the following chapters extensively.

The LOGON framework, on the other hand, produces five different sentences for the same input sentence (listed in (1.2) - (1.6)). The first and second sentences are almost identical to the original sentence. However grammatical they may be, the rest of the sentences are not acceptable paraphrases semantically.

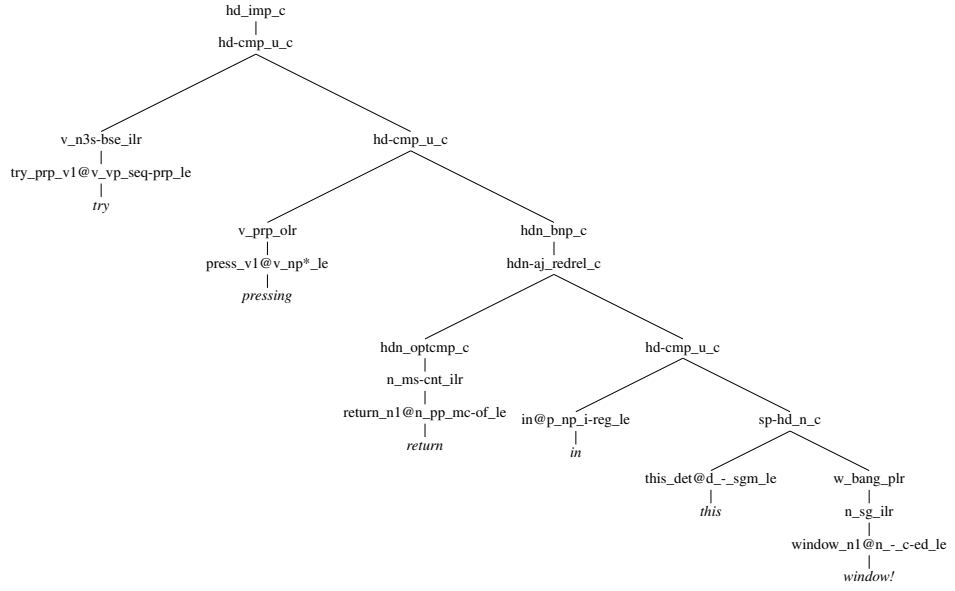
- (1.2) Try pressing return in this window.
- (1.3) Try pressing return, in this window.
- (1.4) Try to be pressing return in this window.
- (1.5) Try to be pressing return, in this window.
- (1.6) Try and be pressing return in this window.

Paraphrases are sometimes inevitable due to the uncertainties in the language itself, not for ambiguities in the manually crafted grammar. Sentences (1.7) - (1.10) are variations generated by XLE for the internal structure depicted in Figure 1.5. These sentences are all happen to be valid paraphrases for the original sentence (Sentence (1.7)) due to the nature of English grammar for adverb disposition.

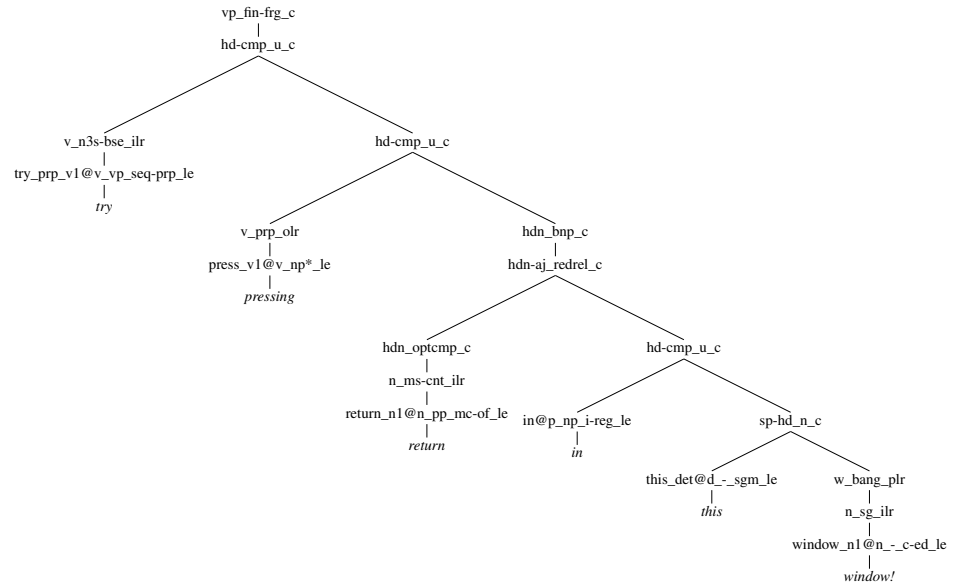
- (1.7) The move significantly expanded Black & Decker's product line but also increased its debt load significantly.



(a)



(b)



(c)

Figure 1.3: Top three parse analyses for Sentence (1.1)

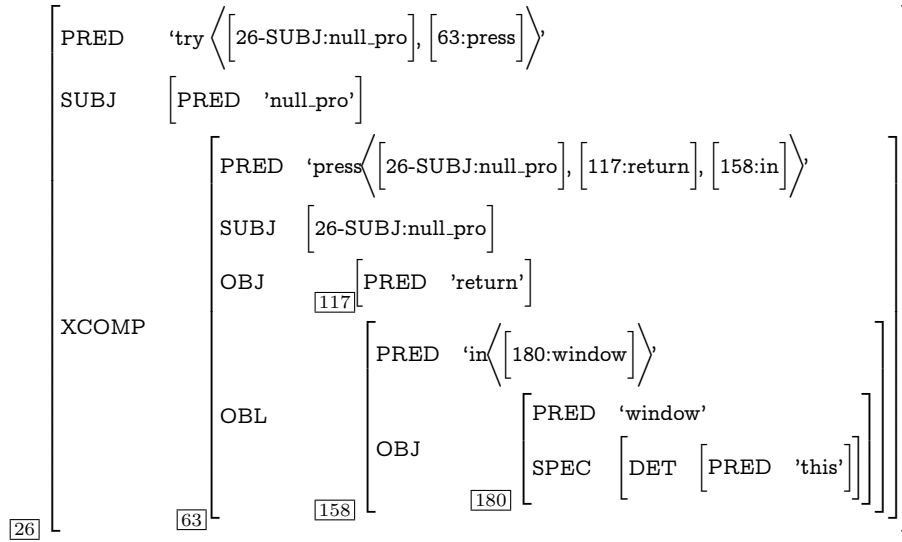


Figure 1.4: XLE representation for Sentence (1.1).

- (1.8) The move significantly expanded Black & Decker's product line but also significantly increased its debt load.
- (1.9) The move expanded Black & Decker's product line significantly but also increased its debt load significantly.
- (1.10) The move expanded Black & Decker's product line significantly but also significantly increased its debt load.

Rankers are thus an essential component of text generators using systems like them since they are responsible for examining the set of alternative generated sentences and ranking them from the most natural to the least. The earliest approach to reranking is using an n-gram language model (LM) (Langkilde and Knight). Due to its limited context (the window size), an LM is not able to capture all the language features for a long string at once. Cahill et al. (2007b) took a grammar writer's perspective by refining the rich manually crafted LFG German grammar to resolve the ambiguity at the grammar level rather than at the level of the generated text for German which is classified as a free-word-order language. They defined various features over the NLG internal structure, similar to the one presented in Figure 1.5. Then by feature engineering, positive or negative contributing features were identified and selected as the most distinguishing features to perform the reranking.

However, not many languages have such fully fledged grammars. What do we do if a rich manually-crafted grammar is not available? Before answering this question, it is worthwhile to clarify what sort of languages are the target of this

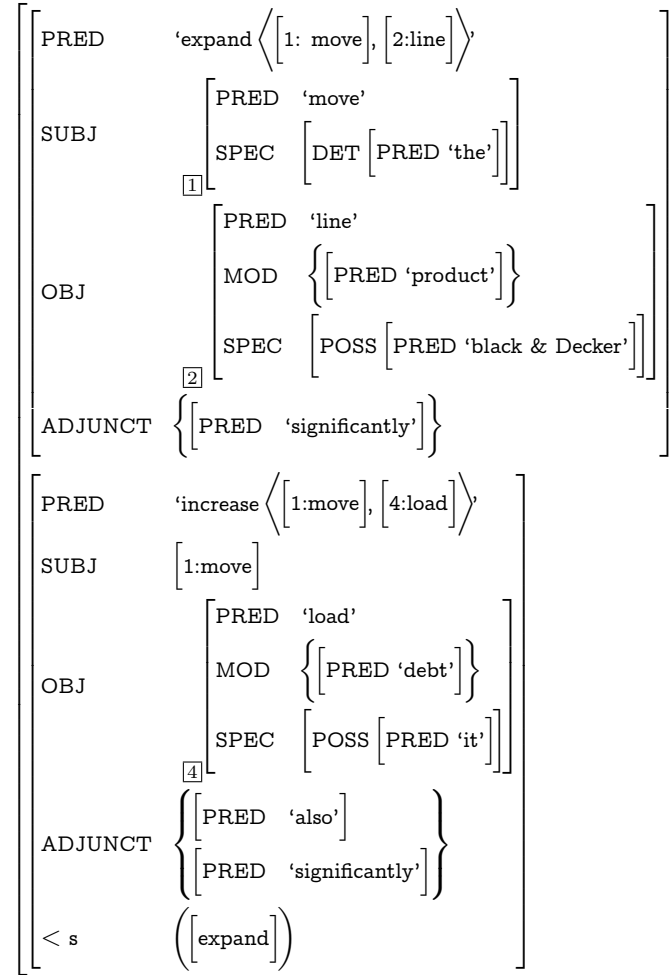


Figure 1.5: XLE internal representation for Sentence 1.7.

research. The focus of NLP research has often been on languages that have sizeable digital resources⁴ such as parallel text, treebanks, semantically tagged corpora, or other annotated resources. These manually crafted resources have then been used to build tools like Part of Speech (PoS) taggers, chunkers and parsers to allow text processing. The rest of the world's languages fall on a wide 'resource availability' cline, from languages like Aatasara of Papua New Guinea with almost zero resources⁵, to Persian, with limited resources⁶. A low-density language in this thesis refers to a language that can be found in the middle of the resource availability cline which might have a small and fragmentary grammar; languages with negligible resources like Aatasara are not within the scope of this thesis. It's quite possible for such languages to face the ranking problem, either because they have a manual grammar that is not elaborate enough or they have other NLP systems that produce text without relying on a grammar. For instance, PEnT1 the English To Persian (Farsi) translator module of PEnTrans (Saedi et al., 2009) has its source language on the rich-end of the cline and its target language is relatively low-density language. PEnT1 implements a rule-based approach of transferring the parse tree from English to a valid parse tree in Farsi. It performs Word Sense Disambiguation (WSD) on the source side using WordNet and picks the mapped sense in FarsNet (Shamsfard, 2008) where the mapping is available⁷. In case the synset in the target language has more than one word, a form of LM derived from Google search is used to determine the final surface realisations. Given the availability of a stand-alone realisation reranker, the quality of the system could have been improved by finding the best sentence amongst the n-best sentences.

Returning to our specific question of interest, we want to investigate whether other sources of structure can be useful in absence of manual grammar: specifically, we will look into various statistical parsers as a source for structural features. In particular, we look at the use of features from unsupervised parsers, as a resource that is available for any language, alone or in combination with features derived from the system grammar as in Cahill et al. (2007b).

1.2 Generation from Bag-of-Words

For manually crafted grammars, as above, the grammar takes care of word ordering. In the absence of manually crafted grammars, a purely data-driven approach

⁴For instance: English, German, French, Chinese, Japanese, and Spanish.

⁵The only resource for Aatasara comes from the An Crúbadán project <http://crubadan.org/writingsystems> which scrapes the web for text from low-resource languages; Aatasara's resources consist of 39 documents of 46437 words in total.

⁶These resources includes a couple of corpora, morphology tools and parsers and listed on https://aclweb.org/aclwiki/Resources_for_Persian

⁷Otherwise, it uses a dictionary and treats all the entries and let the LM decides.

to word ordering is necessary. General purpose word ordering is a subtask of NLG which assigns order to a given bag-of-words. It is considered as a potential candidate for text generation/ re-generation in the absence of manually crafted grammars. There are standard approaches in the literature to address this problem, with the very first being applying the approach of Langkilde and Knight (1998). Regardless of an n-gram language model's good performance on some text, its performance decreases in capturing long dependency between words. For example, if two words, w_1w_2 , have a high bi-gram score, by adding a string of size n in between them, w_1 and w_2 will no longer be in the same n-gram window. It means that we cannot use information on the relative order of these two words.

This shortcoming of LMs in addressing the generation problem suggests the use of some kind of structure as an intermediate representation for a given bag-of-words, so that the word order can be inferred from that with a higher confidence. Considering generation as the inverse of parsing, syntax trees seemed to be a promising structure mainly because data-driven approaches have already proved to be helpful in producing quality parse trees. For instance McDonald et al. (2005b) trained a graph-based parser with a margin-sensitive online training algorithm, Margin Infused Relaxed Algorithm (MIRA), that achieved state-of-the-art performance on Czech and English by capturing useful dependency properties.

Wan et al. (2009) proposed for the first time using structural information in realisation using a graph-based approach. In an inverse of graph-based parsing, the authors mapped the bag-of-words to a weighted graph and applied a Minimum Spanning Tree (MST) algorithm to produce a dependency representation which was then linearised using LM as the final string. This problem has a much higher complexity compared to a typical parsing as there is an exponential number of word choices for the output sentence and each has a factorial number of orderings (Zhang and Clark, 2011). Using a standard spanning tree algorithm, while an improvement over an n-gram language model, produced sentences where for example verbs had unusual number of arguments. Consequently, they proposed an argument satisfaction model to incorporate linguistic features and improve the linguistic validity of the generated tree.

A subsequent approach to the task, proposed by Zhang (2013), uses a search algorithm over various possible input structures ranging from POS-tags to dependency relations, termed as partial-tree linearisation. The search algorithm task looks into two scores to find the best realisation: (1) score of the most probable parse tree for each bag-of-words built in a bottom-up fashion relying on a treebank (2) n-gram language model score.

In keeping with our motivation of minority languages, languages that are mapped somewhere around the middle of the 'resource availability' cline, we try a new approach that is an extension to Wan et al. (2009)'s work. In that work, the authors showed the importance of imposing grammaticality constraints in the process of

generating dependency trees. There are various ways to impose the grammaticality of trees. As proposed by McDonald et al. (2005b), quality dependency trees can be produced using a model trained with MIRA over a dependency tree-bank. Approaches like that of Zhang (2013) hard-code such constraints into the search algorithm. We instead use Integer Linear Programming (ILP), as it allows grammaticality to be implemented via declarative constraints.

Specifically, we use MIRA to learn the dependency weights and local word order by creating a dependency tree and use an LM to decide the final word order. ILP allows the dynamic addition or removal of constraints and MIRA learns the dependency weights from an annotated dependency tree bank. These two main components of the text generator make it a perfect candidate for applying it to various languages. The first component allows the input from the human-expert, e.g, a certain PoS tag should (not) follow another PoS tag. Such constraints can be embedded into the system effortlessly.

1.3 Contribution

The research in this thesis, on surface realisation, tackles two specific tasks: (1) Ranking the output of a text generation system using syntactic features extracted from computational/linguistic resources which is applicable in situations where such resources exists, and (2) Generating text from a bag-of-words. Within each we follow the motivation of making surface realisers one step closer to a generic tool i.e. available to as many languages as possible — especially minority languages — by minimising the reliance on expensive language-specific resources such as grammars and treebanks. The successful reranking results with statistical parsers’ features, leads us to the second part of our research: finding an alternative for crafted grammars deployed in realisers.

With respect to the former part of the research:

1. This thesis shows that a subset of structural features from unsupervised parsers can be as effective as features from supervised parsers where an appropriate feature selection method is used (Chapter 3).
2. This thesis shows the high contribution of features from a manually-crafted grammar to the reranking (Chapter 4).
3. This thesis shows that structural features from supervised parsers are as effective as features from the internal representations of a symbolic computational grammar. (Chapter 4)
4. This thesis shows that the effect of internal versus external parse features varies depending on the size of the grammar. (Chapter 4).

The findings in respect to the latter part of the research are:

5. This thesis introduces a framework for surface realisation which is specifically useful for languages that lack manually-crafted-grammar or supervised dependency parsers using a combination of statistical and declarative approaches. We present an optimisation-based method that serves as the search algorithm to assign an intermediate structure to a given bag-of-words to guide the realisation. This framework benefits from flexibility in adding/removing structural/linguistic constraints. This last specification is specially valuable when it comes to adapting the realiser to a new language (Chapter 5).
6. This thesis shows that incremental addition of constraints makes the linearisation process more efficient. Also the flexibility to incorporate richer linguistic constraints produces a better generated text. (Chapter 6)
7. This thesis confirms that languages with a treebank can benefit from MIRA to learn the assignment of intermediate structure for linearisation purposes. The accuracy of linearised strings learned by this algorithm beats those produced by Maximum Likelihood Estimate (MLE)-based baselines. (Chapter 6)
8. In this thesis, we show that some of the language-specific properties are producing a better linearisation if they are applied to the problem as soft constraints rather than hard constraints.(Chapter 6).

1.4 Thesis Outline

This thesis is divided into seven chapters.

In Chapter 2, we review the existing literature on NLG systems focusing on the most agreed upon architecture and generators that deploy symbolic grammars. We provide an overview of surface realisation ranking and surface realisation and drill down to works that are the basis of this research. We finish this chapter with a brief discussion of the creative RNN-based text generation systems, which have received a lot of attention recently, and why we are not investigating them in this work.

In Chapter 3, we introduce the framework, taken from Cahill et al. (2007b), for statistical reranking of grammar output and for evaluating reranking accuracy. We then reproduce Cahill’s approach to using grammar-derived features, and then we introduce several models to investigate how sources of syntactic information external to the grammar — specifically, supervised and unsupervised parsers — can contribute to reranking.

In Chapter 4, we investigate how features from statistical parsers investigated in Chapter 3 can be combined with features from manually crafted grammars.

Also, we compare the contribution of various features from two manually crafted grammars that differ in scale.

In Chapter 5, we replicate Wan et al. (2009)’s work using an ILP approach and further extending it by adding flexible grammaticality conditions in the form of ILP constraints.

In Chapter 6, we use MIRA rather than MLE for assigning a dependency tree that represents the induced syntactic structure for the generated sentence.

In chapter 7, we summarise the contribution of this thesis and suggest possible further research.

Chapter 2

Literature Review

In this chapter we review various aspect of Natural Language Generation (NLG) literature starting for context with the older work that evolved into the ‘consensus architecture’, then moving to statistical approaches, then focusing on the two tasks noted in Chapter 1, reranking and generating from a bag-of-words.

2.1 Overview of NLG Systems

NLG is the subfield of Artificial Intelligence and Computational Linguistics that refers to the production of meaningful text from a non-linguistic representation of information (like knowledge bases) using computer systems. Such systems basically require combining language knowledge and application domain knowledge to perform the generation task (Reiter and Dale, 2000).

Although the above definition seems to be a standard one, Evans et al. (2002) in “What is NLG” discussed NLG systems’ characteristics more in depth and tried to give a more exact definition for them. They start the discussion with the necessity of having an agreed view about what is the starting point of such systems, including: the kind of input, the assumptions they make about it and what they do with it before turning it into some kind of textual output. They proposed two principles characterizing NLG system:

PRINCIPLE A: NLG is a *linguistic* manipulation of data.

PRINCIPLE B: NLG manipulates (linguistically) deeper information to produce shallower information.

The first principle put the emphasis on linguistic manipulation. So it should involve language-specific resources or the manipulation itself should be language specific. On the other hand, the second one insists on change in the depth of information.

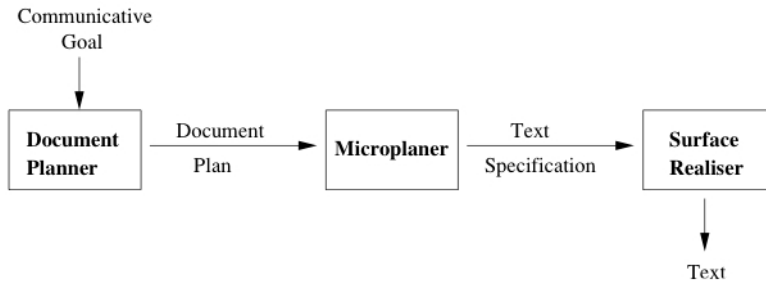


Figure 2.1: An NLG system architecture. (Reiter and Dale, 2000)

<i>Module</i>	<i>Content Task</i>	<i>Structure Task</i>
Document Planner	Content Determination	Document Structuring
Microplaner	Lexicalisation Referring expression; Generation	Aggregation
Surface Realiser	Linguistic Realisation	Structure Realisation

Table 2.1: The distribution of content and structure tasks per module in ‘consensus architecture’, reproduced from Figure 3.1 from Reiter and Dale (2000)

The authors assert that summarisation or translation don’t change the information level. These just represent the information another way.

Following these two principles and two conditions that clarify linguistic operation, the boundary of NLG systems narrowed down and some of systems known as NLG would be just a summariser that maps from a more to a less detailed language representation at the same “depth”. For example they consider WEATHERREPORTER, the system used as case study through the ‘Building Natural Language Generation Systems’ book by Reiter and Dale (2000). They discuss a subtask in that system which is responsible for summarisation of a month’s worth of information. Evans et al. declare this task cannot be regarded as an NLG task based on their given principles. They assert if this step is non-linguistic, then it is ruled out by principle A; if it is linguistic then it is ruled out by principle B. However since, Reiter and Dale (2000)’s definition of NLG is the most widely accepted in the field, we refer to this definition of NLG from this point onward in the thesis.

The most widely accepted architecture is the pipeline architecture, also proposed by Reiter and Dale (2000). It is generally known as the ‘Consensus architecture’ and has three principal modules: 1. Document Planner, 2. Microplaner and 3. Surface Realiser. This pipeline is depicted in Figure 2.1. These three modules are responsible for two major groups of generation tasks, Content Tasks and Structure. Table 2.1 represents each module along with its two major tasks.

Although the architecture mentioned is the most widely agreed upon architecture,

Cahill et al. (1999) posed the following question:

“... whether Reiter’s ‘Consensus NL Generation Architecture’ really exists, and if so whether it is a suitable candidate for a ‘reference architecture’ for NLG systems. Our answer to the first question is a tentative yes, but we are less comfortable to accept the second.”

The above search for reference architecture led to RAGS (Reference Architecture for Generation Systems) which is a specification of an abstract NLG system architecture to support sharing, re-use, comparison and evaluation of NLG technologies (Mellish et al., 2006).

2.1.1 NLG systems

The architectures discussed above are idealised architectures. Practical systems may vary in many ways from these; below, we review some platforms that can be used to develop practical systems. D2S (Theune et al., 2001) is designed to generate speech from data. It has been used for different languages like English, German, Dutch and also for different domains like music and travel. FUF (Goldberg et al., 1994), another platform, is actually an interpreter for functional unification grammars and it is used for NLG systems like PLANDOC (McKeown et al., 1994) which is implemented for English and helps telephone network engineers to document their simulation results.

NLG systems are able to generate text from different kinds of input. For example ModelExplainer (Lavoie and Rambow, 1997) produces text based on object-oriented class models. Data-driven NLG is a category of generation systems that produces text based on raw data saved in a knowledge-base. Such systems can be very useful in situations where a human author frequently writes text from data, e.g. reports on weather forecasts. The author should not only write the text but also should decide how to analyse the data. Such a system consumes the raw data directly, for example the amount of rain in a specific day. Additionally, the system should decide where more analysis is required prior to generation, so it can clarify the discussion or even make meaningful comparisons like calculating the average rainfall in a given month or indicating temperature in comparison to the same day of the previous year. This statistical analysis of input data can produce precise reports without human intervention.

Data-Driven systems FOG (Goldberg et al., 1994) is a bilingual system that produces text in English and French that assists meteorologists with composing weather forecasts. The data is collected from weather stations and saved into a database. Some of the complexities that the system faced included deciding on the degree of details the reports cover, or where to use inexact temporal terms or more

precise phrases, e.g. late this evening vs Friday midnight. FOG’s sample report and input was previously provided in Figure 1.1 and Figure 1.2.

Another data-driven text generating system is SPOTLIGHT (Anand and Kahn, 1992), a knowledge-based market analyzer. This system is responsible for converting large amounts of data into five brief, clearly understandable reports. These reports assist both manufacturers and retailers in tracking the scale and movement of their products, assess the effectiveness of promotional strategies, and compare the performance of competing products and product segments. Writing reports on such a great amount of data is not only time consuming and error-prone, but also it might need a domain expert to revise. This multilingual system generates a language-independent representation of data, then maps it to each language separately by deploying a grammar and a dictionary. André et al. (2000) and Nijholt et al. (2003) generated real-time commentary on RoboCup simulation league games. Nijholt et al. compared three real-time commentary systems. As all these systems concentrate on RoboCup Simulator league the match is not visually observed, therefore they obtain the input data from the software that monitors the game. The input data consists of (1) information describing a player including its location and orientation and (2) location of the ball and game information like score and goal kicks. The output of all these systems are in speech format.

Note that due to the complexity of the language and demand for highly reliable end product, commercial NLG systems — such as Arria NLG¹, NarrativeScience² and Yesop³ — are heavily reliant on data-driven approaches that guarantee the quality of the generated text.

2.2 Data-Driven Methods: Surface Realisation Ranking

According to the Reiter and Dale (2000) architecture for NLG systems, a system consists of three components: Document planner, Microplaner and Surface realiser. The realiser converts an abstract representations of sentences or semantics into natural language sentences. Such a realiser can also be found in other Natural Language Processing (NLP) applications like Machine Translation or Text Summarisation. The main issue with this sort of realiser⁴ is that it typically generates multiple sentences that are sometimes paraphrases; some of them are equivalent

¹<https://www.arria.com/>

²<https://www.narrativescience.com/>

³<http://yseop.com/EN/smart-business-intelligence>

⁴Another type of realiser is a symbolic realiser that produces only one sentence for a given input semantic representation.

but the rest of them are misleading. In Figure 2.2 a sample of realiser output is shown.

In fact, the reranking process can be regarded as the reverse action of parse selection. The output of a parser, a parse forest, includes all the possible parse trees for a given sentence. Similarly, a generation forest (the output of a text generator) contains one or only a few acceptable structures in addition to other similar structures that are less preferred. Ranking the output of the realiser, the system must sort the generated sentences so the most relevant representation comes first and so on while for parse selection, it should choose the n-best parses from numerous parses given for one sentence and put them in order so the best comes first.

1. remember that dogs must be on a leash.
2. remember dogs must be on a leash.
3. on a leash remember that dogs must be.
4. on a leash remember dogs must be.
5. a leash remember that dogs must be on.
6. a leash remember dogs must be on.
7. dogs remember must be on a leash.

Figure 2.2: A sets of a generated sentences using LinGO ERG. (Velldal and Oepen, 2005)

There are different approaches to perform this task and like many other NLP applications, statistical approaches to surface realisation have shown significant improvements over non-statistical approaches. Traditionally, realisers refer to a corpus to build a model for the language. In that way the best word order can be chosen by consulting that model.

Langkilde-Geary (2000) introduced the idea of statistical realisation ranking by using a ‘lattice’, a graph where each arc is labelled with one word, to represent corpus-based knowledge, like common phrases and decisions. There were some drawbacks like unavoidable duplication, and the independence between many choices cannot be fully exploited. She consequently improved her model by converting the lattice to a forest so that a group of nodes that occurs more than once is given a new and unique label. She proposed a bottom-up dynamic programming algorithm for ranking the forest by comparing alternate phrases corresponding to the same semantic input. She considers two different types of score for each phrase in the forest: Internal (context-independent) and external (context-dependent). The internal is computed once and saved with the phrase while the external node is computed by considering the phrase siblings. Equation 2.1 calculates the internal

score for the given phrase p recursively.

$$I(p) = \prod_{j=1}^J I(c_j) * E(c_j | context(c_1..c_j - 1)) \quad (2.1)$$

where I is internal score and, E is the external score, and c_j is a child node of p . Both I and E and $context$ formulation depend on a language model so if the language model is bigram-based, $I(c_j)$ would be equal to 1 for all c_j and E can be calculated as in Equation (2.2).

$$E = P(FirstWord(c_j) | LastWord(c_j - 1)) \quad (2.2)$$

It is also possible to make a more complex language model by including features such as head word, part-of-speech tag, constituent category, etc. After the score for each phrase is computed, the ranking algorithm tries to concatenate phrases in a way the maximum score is delivered. This ranker was used in a general-purpose sentence generator system introduced by Langkilde-Geary (2002).

Ratnaparkhi (2000) presented three trainable systems for surface natural language generation (NLG1, NLG2 and NLG3) to describe flights in the domain of air travel. All of these three systems deploy annotated corpora with domain-specific attributes and follow a two step procedure depicted in Figure 2.3. The third system also require a corpus with syntactic dependency information annotated. NLG1 uses phrase frequency to generate a whole phrase in one step i.e the most frequent template in the training data that corresponds to a given set of attributes is chosen. The other two deploy two different maximum entropy probability models to individually generate each word in the phrase. The main difference between these systems is that NLG3 tries to overcome the shortcomings of NLG2 in predicting the next word. So it uses conditioning on syntactically related words to generate more accurate sentences. It incorporates conditions on the parent and the two closest siblings and the direction of child relative to parent. It also requires the corpora to be annotated with tree structure like the sample dependency tree shown in Figure 2.4. Both NLG1 and NLG2 perform feature selection followed by a search for the best tree.

2.2.1 Realisation Ranking using Log-linear Models

A major advance in statistical ranking approaches was using log-linear models. Log-linear models aim to describe language by giving a weighted aggregation of arbitrary features; they can be seen as a generalisation of simple language models. So, they can be used in other NLP applications such as parse selection and Machine Translation result ranking. Early works on log-linear models used just language models as features. However, since then new features derived from both language

Input to Step 1: { \$city-fr, \$city-to, \$time-dep, \$date-dep }

Output of Step 1: “a flight to \$city-to that departs from \$city-fr at \$time-dep on \$date-dep”

Input to Step 2: “a flight to \$city-to that departs from \$city-fr at \$time-dep on \$date-dep”,
 { \$city-fr = New York City, \$city-to = Seattle, \$time-dep = 6 a.m., \$date-dep = Wednesday }

Output of Step 2: “a flight to Seattle that departs from New York City at 6 a.m. on Wednesday”

Figure 2.3: Two steps of NLG process for NLG1, NLG2 and NLG3 with corresponding input and output. Words starting with \$ are attributes. (Ratnaparkhi, 2000)

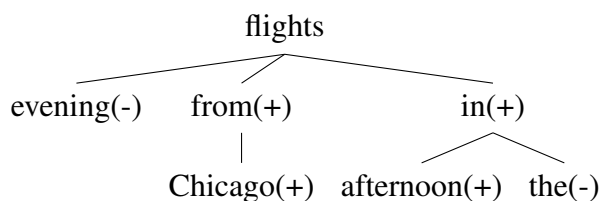
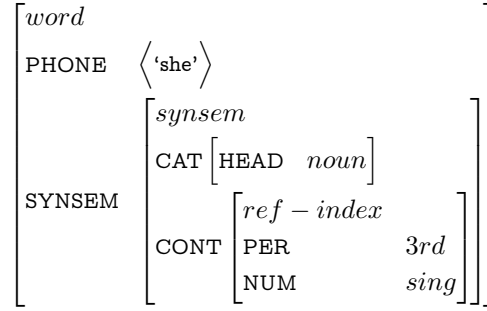
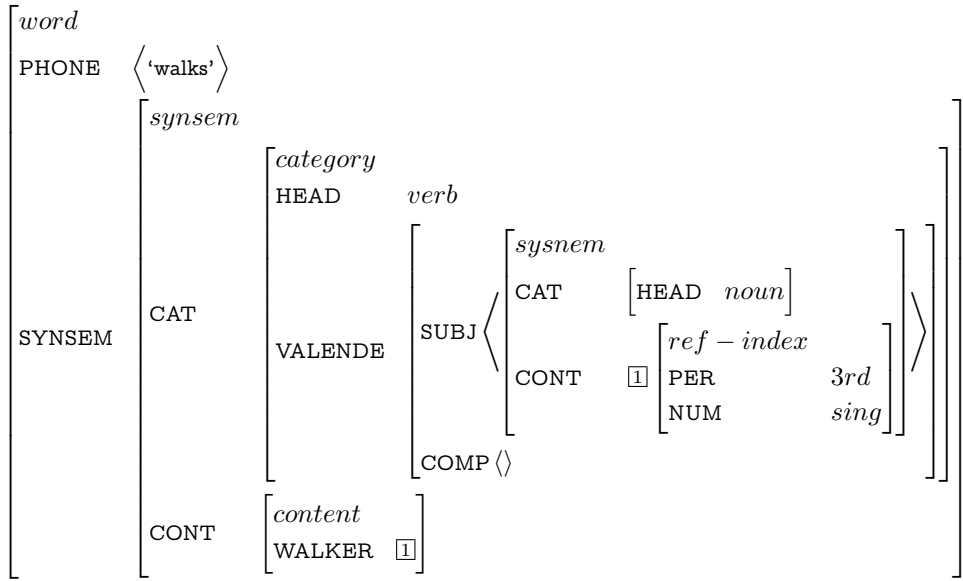


Figure 2.4: Sample dependency tree for the phrase *evening flights from Chicago in the afternoon* + and - signs indicate right and left child, respectively.(Ratnaparkhi, 2000)

model scores and features related to syntactic structure are employed. Here we give an overview of two of the most relevant research approaches that are the base of realisation ranking in this thesis: both of these apply the log-linear models to the output of the symbolic grammar-based generators. We briefly discuss the grammar formalisms and resources that they used prior to the methodology discussion. We discuss the latter research in more detail as it is the basis of the work presented in this thesis.

2.2.1.1 Realisation Ranking using Log-linear with Head-driven Phrase Structure Grammar (HPSG)

HPSG HPSG is one of the major formalisms for syntactic theory. Being precisely formalised, HPSG has become one of the more popular formalisms for sentence analysis and generation. HPSG grammars are expected to have a set of base

Figure 2.5: HPSG feature structure for pronoun *she*Figure 2.6: HPSG feature structure for pronoun *walks*

words which would be extended dramatically by applying a set of lexical rules. This formalism uses an Attribute Value Matrix (AVM) to represent various word attributes such as PHON(ological) and SYNSEM (syntactic, and semantic) such as word category, with what other words it must appear and what level in the tree the node is. Figures 2.5 and 2.6 are AVMs for pronoun *she* and verb *walks*. As per the annotation in Figure 2.5, *walks* is a verb that requires a subject and no complementisers. As specified by its SUBJ functional structure (f-structure), the appropriate subject must be a noun, and be third-person singular. Comparing the expected subject f-structure with the one that describes *she*, it can be concluded that *she* is the appropriate subject for the verb *walks*. This is how this formalism enforces valid syntax and word order.

The LinGO Redwoods treebank⁵ is a rich and dynamic treebank and is based on open-source HPSG resources. This treebank is a collection of hand-annotated corpora analysed with the LinGO English Resource Grammar (ERG)⁶. LinGo Redwoods allows retrieval of linguistic data in varying granularity and in the constant evolution and regular updating of the treebank itself (Oepen et al., 2002). This treebank also allows alternative analyses including dispreferred ones per utterance.

Velldal and Oepen (2005) implemented realisation ranking for LOGON, a machine translation system with an NLG component that generates English sentences from a semantic representation. They employ the Lingo-ERG for the generation task.

The authors applied three different statistical models in their realiser, evaluating them against each other: an n-gram language model, a discriminative maximum entropy model using structural features, and a combination of these two. They produced a ‘symmetric treebank’ for both training and evaluation. The notion of symmetric treebank was first introduced by Velldal et al. (2004) and it is composed of:

1. a set of pairings of surface forms
2. a set of alternative analyses for each surface form, and
3. sets of alternate realisations of the semantics.

The preferred realisations are automatically specified by comparing the yields of the generated trees with original strings in a treebank. The authors employed the Redwoods treebank then applied re-generation followed by the previously defined process as an alignment method to create the symmetric treebank. Negative log-probabilities were employed to rank the n-gram model. For the maximum entropy ranker they use a conditional log-linear model which calculates the probability of realisation r given semantic representation s . In the log-linear model described by equation 2.3, f is a set of real-valued feature functions that describe properties of the data, and an associated set of learned weights λ that determine the contribution of each feature.

$$p_{\lambda}(r|s) = \frac{1}{Z_{\lambda}(s)} q(r|s) \exp\left(\sum_{i=1}^d \lambda_i f_i(r)\right) \quad (2.3)$$

⁵<http://moin.delph-in.net/RedwoodsTop>

⁶<http://www.delph-in.net/erg/>

where $Z_\lambda(s)$ is the normalisation term defined as

$$Z_\lambda(s) = \sum_{r' \in Y(s)} q(r'|s) \exp\left(\sum_{i=1}^d \lambda_i f_i(r)\right) \quad (2.4)$$

and $Y(s)$ gives the set of all possible realisations of s . The so-called reference or default distribution q is often only implicit and defined as $\frac{1}{Y(s)}$. It can also be substituted by some other reference distribution to incorporate prior knowledge in the model.

2.2.1.2 Realisation Ranking using Log-linear with Lexical Functional Grammar (LFG)

This is a very brief overview focussing on relevant characteristics for this thesis; for a more complete survey of the formalism, see Dalrymple (2001) or Bresnan (2000).

LFG The LFG formalism represents the structure of syntactic constituents, constituent structure (c-structure) and the grammatical functions, f-structure, by trees and AVMs, respectively. These two primary structures of LFG are systematically linked. Figure 2.7 and 2.8 provides the respective analyses for sentence (2.1).

(2.1) Pierre Vinken, former Elsevier chairman, died at age 83.

Attributes in an f-structure can be either complex (embedding one or more attributes) or atomic (containing a single value). An example of a complex attribute is SUBJ. it represents the subject of a particular predicate and accepts multiple attributes in the form of another AVM, as its value. Another example is ADJUNCT; since English has no limit on the number of the modifiers a noun can take, ADJUNCT is defined as a complex attribute that accepts a set of attribute-value pairs (Dalrymple, 2001). As can be seen ADJUNCT occurred 4 times and has a set as its value (annotated as 97, 106, 243 and 303 in Figure 2.8)⁷. Unlike complex attributes, atomic attributes accept only a value from a set of available atomic values. For instance NUM (number) and GEN (gender) can take a value from {SING/DUAL/PL} and {FEM/MASC}, respectively. These concepts will be used later in Section 4.2.2 to describe f-structure feature templates.

The Xerox Linguistics Environment (XLE) provides a platform for parsing and generating Lexical Functional Grammars (LFGs) that comes with a rich graphical

⁷These numbers are XLE annotations to uniquely identify attribute values

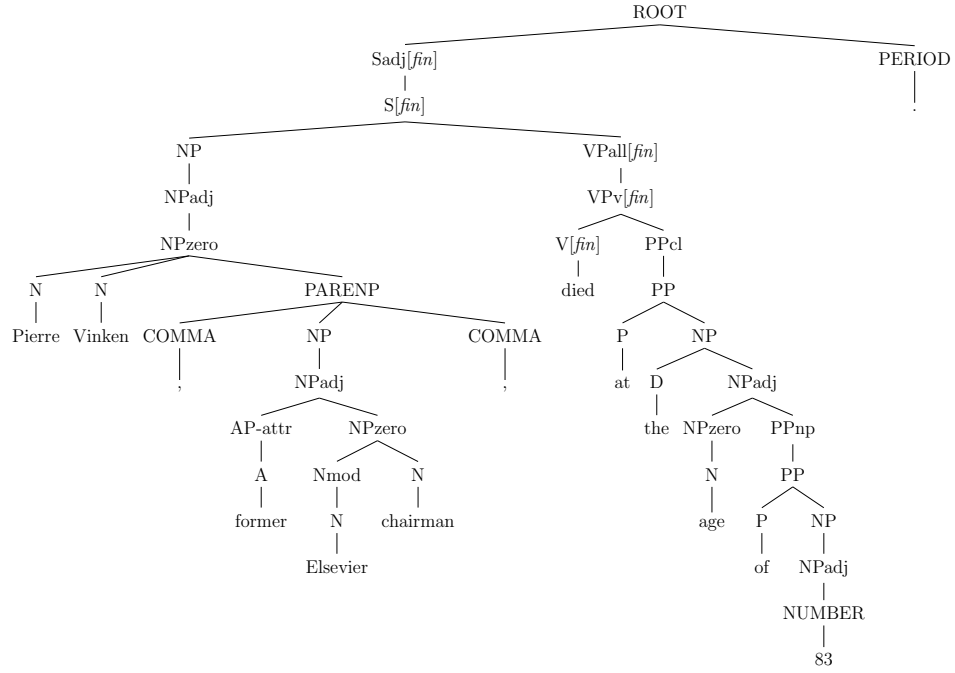


Figure 2.7: c-structure for sentence (2.1)

user interface for writing and debugging such grammars. It is the basis for the Parallel Grammar Project, which is developing industrial-strength grammars for Arabic, Chinese, English, French, German, Georgian, Hungarian, Indonesian, Irish, Japanese, Malagasy, Murrinh-Patha, Norwegian, Polish, Spanish, Tigrinya, Turkish, Urdu, Welsh and Wolof (Sulger et al., 2013). In fact, all the LFG analyses in this thesis were produced by XLE.

German LFG Grammar German has a reversible broad-coverage LFG grammar that was initially developed as part of the ParGram project (Butt et al., 2002, Dipper, 2003) with the initial focus on phenomena discussed in theoretical syntax. Later, Rohrer and Forst (2006) improved the coverage benefiting from techniques that induce grammars from treebanks. As reported by authors, this grammar had 274 LFG style rules, which compiled into an automaton with 6,584 states and 22,241 arcs. This broad-coverage grammar is also reversible i.e. it can be used for both parsing (producing c-structure and f-structure from a given sentence) and generation (conversion of surface realisations from of a well-formed input f-structure into a sentence). This grammar also provides an acceptable parsing coverage of about 80% in terms of full parses on newspaper text and it falls back to a FRAGMENT grammar for the out-of-coverage sentences and provides a partial analyses for them. The FRAGMENT grammar as defined by (Riezler et al., 2002, p. 1) is a grammar that is deployed in the absence of a complete parse and allows the input

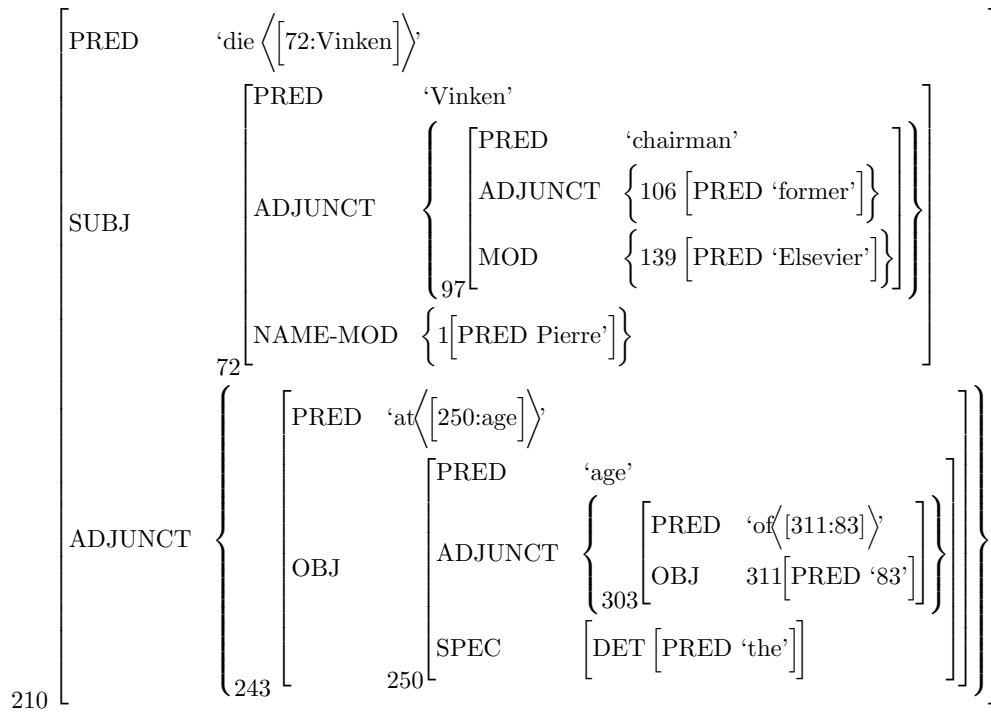


Figure 2.8: f-structure for sentences (2.1)

to be analysed as a sequence of well-formed chunks. The set of fragment parses is then chosen on the basis of the fewest-chunk method. With this combination of full and partial parsing techniques 100% grammar coverage is achieved on unseen data.

We believe that not many languages have such fully fledged LFG grammar as English, German, Norwegian, Japanese, Arabic, and Urdu to assist language processing and generation. Other languages either have no computational grammars or they are in the process of development which means it cannot be as comprehensive as the ones listed above. To name a few, LFG grammars for Greek, Irish and Brazilian-Portuguese are under-development.

Parse Tree Disambiguation Disambiguation within parsing refers to the problem of choosing which subtree should be present in the final parse tree where there is more than one way of expanding a given node so that the final tree structure represents the given sentence the best. This is exactly the reverse task for surface realisation: assigning the best surface text to a given structure. Riezler et al. (2002), Riezler and Vasserman (2004) and Forst (2007) approached the parsing problem by training a model over a set of features that represent preferred and dispreferred parse trees for a given sentence. Riezler et al. (2002) and Riezler and Vasserman (2004) found the use of c-structure and f-structure in ranking of generated text to be helpful in parse tree disambiguation. Riezler et al. (2002) proposed a log-linear model for LFG parse disambiguation for English. They defined a set of simple, mostly locally restricted c-structure and f-structure feature templates, as listed in Table 2.2. However, such features were claimed not to be as efficient in parse tree disambiguation for free word order languages such as Japanese and German (Cahill et al., 2007b). Forst (2007) showed that a significant improvement can be achieved for parse tree disambiguation in German by extending Riezler et al.’s templates in order to capture more of the linguistic aspects of the sentences. Such linguistically motivated features capture language-specific characteristics such as linear order of grammatical functions, the (surface and functional uncertainty path) distance of an extraposed constituent to its f-structure head, the nature of a DP in relation to its grammatical function (pronominal vs. full DP, animate vs. inanimate).

Reranking XLE output Cahill et al. (2007b,a) were inspired by this approach and looked into the usefulness of parse disambiguation features in the reranking task. They presented a ranking model that learns the association between a set of structural features and the preferred and dispreferred surface realisation for German. They used the XLE framework with the above mentioned German reversible broad-coverage LFG grammar.

They proposed deploying Forst’s extended feature templates and studied to what extent they can improve the realisation ranking task for German, considering the

Table 2.2: Features used by Riezler et al. (2002), Riezler and Vasserman (2004) for the disambiguation of English ParGram LFG parses (Cahill et al., 2007a)

feature template and parameters	Explanation
fs_attr <attrs>	counts number of occurrences of attribute(s) <attrs> in the f-structure
cs_label <cat>	counts number of occurrences of category <cat> in the c-structure
fs_attr_val <attr> <val>	counts number of times f-structure attribute <attr> has value <val>
cs_num_children <cat>	counts number of children of all nodes of category <cat>
fs_adj_attr <attr1> <attr2>	counts the number of times feature <attr2> is immediately embedded in feature <attr1>
fs_sub_attr <attr1> <attr2>	counts the number of times feature <attr2> is embedded somewhere in <attr1>
cs_adjacent_label <cat1> <cat2>	counts the number of <cat1> nodes that immediately dominate <cat2> nodes
cs_sub_label <cat1> <cat2>	counts the number of <cat1> nodes that (not necessarily immediately) dominate <cat2> nodes
cs_embedded <cat><Depth>	counts the number of <cat> nodes that dominate (at least) <Depth> other <cat> nodes
cs_conj_nonpar <Depth>	counts the number of coordinated c-structures that are not parallel at <Depth> levels under the coordinated constituent
lex_subcat <Lemma><SCFs>	counts the number of times <Lemma> occurs with one of the subcategorisation frames in <SCFs>
Additional Linguistically Motivated Features	
ADD-PROP MOD1 <Lemma>	counts the number of times a given lemma occurs as a member of a MOD set
ADD-PROP F2 <Lemma> <PoS>	counts the number of times a given lemma occurs as a particular <PoS>
ADD-PROP ACTIVE/PASSIVE <Lemma>	counts the number of times a (verb) lemma occurs in active/passive voice
ADD-PROP isCommon/Def/Pronoun/... <GF>	determines whether a DP with function <GF> is common, definite, pronominal, etc.
ADD-PROP DEP11 <PoS1> <Dep> <PoS2> <PoS2>	counts the number of times a sub-f-structure of type is embedded into a (sub-)f-structure of type <PoS1> as its <Dep>
ADD-PROP PATH	counts given instantiations of functional uncertainty paths
ADD-PROP PRECEDES <GF1> <GF2>	counts the number of times a <GF1> precedes a <GF2> of the same (sub-)f-structure
DISTANCE-TO-ANTECEDENT %X	distance between a relative clause and its antecedent
ADD-PROP DEP12 <PoS1> <Dep> <PoS2> <Lemma2>	counts the number of times a sub-f-structure of type <PoS2> and with <Lemma2> as its PRED is embedded into a (sub-)f-structure of type
ADD-PROP DEP21 <PoS1> <Lemma1><Dep> <PoS2> <PoS2>	counts the number of times a sub-f-structure of type is embedded as its <Dep> into a (sub-)f-structure of type <PoS1> and with <Lemma1> as its PRED
ADD-PROP PRECEDES <Lemma> <GF1><GF2>	counts the number of times a <GF1> subcategorised for by a PRED <Lemma> precedes a <GF2> subcategorised for by the same PRED
ADD-PROP MOD2 <Lemma1> <Lemma2>	counts the number of times <Lemma2> occurs in the MOD set of a (sub-)f-structure with <Lemma1> as its PRED
ADD-PROP VADJUNCT PRECEDES <Prep1> <Prep2>	counts the numbers of times an ADJUNCT PP headed by <Prep1> precedes an ADJUNCT PP headed by <Prep2>, both being in an f-structure with a VTYPE
ADD-PROP DEP22 <PoS1> <Lemma1> <Dep> <PoS2> <Lemma2> <PoS2>	counts the number of times a sub-f-structure of type and with <Lemma2> as its PRED is embedded as its <Dep> into a <Dep> into a <Dep> into a (sub-)f-structure of type <PoS1> and with <Lemma1> as its PRED

fact that not all the structural features that contribute to parse disambiguation can promote reranking precision and not all features that captures surface properties are present in that template set. For example in realisation ranking lexical dependencies are given, therefore features that capture lexical dependencies would not help. On the other hand, realisation ranking requires more surface features such as word order.

They used a log linear model (introduced on page 20) similar to Velldal and Oepen (2005)'s (previously discussed on page 23) with two major differences: (1) they used the LFG formalism instead of HPSG, and (2) they switched the language from English to German which is a non-configurational language. They used the German reversible broad-coverage LFG grammar to regenerate alternatives from a unique f-structure. They applied a ranking algorithm to the alternatives. For example, assume the f-structure illustrated in Figure 2.9 is the only analysis produced by XLE for sentence (2.2) and the generator produced multiple realisations (Figure 2.10) these multiple realisations would be then passed to the reranker to be ordered in terms of goodness.

(2.2) Die Nato werde nicht von der EU geführt.

The NATO is not from the EU led.

NATO is not led by the EU.

Their log-linear model was built upon three distinct categories of features: language model features (LM), c-structure features (CF) and additional features (AF).

Features in the first category take advantage of the language model (LM) which is a well-established resource to address stochastic realisation ranking (Langkilde and Knight). A trigram language model score for each realised string and the length of the string, i.e, number of the words, are the two deployed LM features.

However, using it on its own does not always produce good results; i.e there cases in which the original sentence is not amongst the 5-best sentences after reranking. They thus incorporated the language model score associated with each alternative string as just one feature in their model.

CFs, as the name suggests, were extracted from c-structure. Such features can include the number of times a particular category label occurs in a given c-structure, or the number of times one particular category label dominates another.

They listed the information that can be encapsulated in f-structure features as follows:

Linear order of functions SUBJ generally precedes OBJ,

Adjunct position sentence beginning, distance from the verb, etc.

Partial VP fronting generally marked and thus dispreferred

Die Nato werde von der EU nicht geführt.
 Die Nato werde nicht von der EU geführt.
 Nicht von der EU geführt werde die Nato.
 Nicht werde von der EU die Nato geführt.
 Nicht werde die Nato von der EU geführt.
 Nicht geführt werde von der EU die Nato.
 Nicht geführt werde die Nato von der EU.
 Von der EU nicht geführt werde die Nato.
 Von der EU werde die Nato nicht geführt.
 Von der EU werde nicht die Nato geführt.
 Von der EU geführt werde nicht die Nato.
 Von der EU geführt werde die Nato nicht.
 Geführt werde die Nato nicht von der EU.
 Geführt werde die Nato von der EU nicht.
 Geführt werde nicht von der EU die Nato.
 Geführt werde nicht die Nato von der EU.
 Geführt werde von der EU nicht die Nato.
 Geführt werde von der EU die Nato nicht.

Figure 2.10: Alternative realisations for the f-structure in Figure 2.9 generated by XLE.

Regardless of the potential contributions of f-structure features, they had to be combined with other feature types; otherwise all the alternatives would get identical feature set due to the fact that they were created from a unique f-structure. Features in the AF category were expected to contribute to the ranking problem as it captures more linguistic data by combining the information from both intermediate structures. An example of such a feature is the number of children the nodes of a particular category have.

They construct a list of 186,731 features for training a log-linear model using the templates defined by Riezler et al. (2002), Riezler and Vasserman (2004) and Forst (2007). Out of these, only 1,471 actually occur in their training data. They used the `cometc` software⁸ to carry out feature selection using incremental feature selection and l_1 regularisation presented in Riezler and Vasserman (2004).

As a consequence of this selection, only 360 features were used for reranking the alternative solutions. These feature templates along with their description are listed in Table 2.3.

For training, they built a symmetric treebank of 8,609 packed c/f-structure repre-

⁸`cometc` provided with the XLE platform and performs maximum likelihood on standardised feature values and offers several regularisation and/or feature selection techniques.

Table 2.3: Features templates used by Cahill et al. (2007b) for semi-automatic feature construction for parse disambiguation (Cahill et al., 2007b)

Name of feature template and parameters	Explanation
C-structure Features	
cs label <cat>	counts number of occurrences of category <cat> in the c-structure
cs right branch	counts number of right children
cs num children <cat>	counts number of children of all nodes of category <cat>
cs adjacent label <cat1> <cat2>	counts the number of <cat1> nodes that immediately dominate <cat2> nodes
cs sub label <cat1> <cat2>	counts the number of <cat1> nodes that (not necessarily immediately) dominate <cat2> nodes
cs embedded <cat> <Depth>	counts the number of <cat> nodes that dominate (at least) <Depth> other <cat> nodes
cs conj nonpar <Depth>	counts the number of coordinated c-structures that are not parallel at <Depth> levels under the coordinated constituent
Additional Linguistically Motivated Features	
ADD-PROP PATH	counts given instantiations of functional uncertainty paths
ADD-PROP PRECEDES <GF1> <GF2>	counts the number of times a <GF1> precedes a <GF2> of the same (sub-)f-structure
ADD-PROP PRECEDES <Lemma> <GF1>	counts the number of times a <GF1> subcategorised <GF2> for by a PRED <Lemma> precedes a <GF2> subcategorised for by the same PRED
DISTANCE-TO-ANTECEDENT %X	distance between a relative clause and its antecedent
Language Model Features	
GEN NGRAM SCORE %X	3-gram language model score assigned to the generated sentence
GEN WORD COUNT %X	number of words in the generate

sentations in a similar to Velldal et al. (2004). They first excluded the structures for which only one string was generated, since the log-linear model needs a pair consisting of a preferred and a dispreferred sample to learn from at a given point in the training. Then they followed the steps below and prepared the symmetric treebank (Cahill et al., 2007b, p. 11):

1. Parse the input sentence from the TIGER Treebank using XLE.
2. Keep only the analyses that are compatible with the TIGER Treebank annotation.
3. Of all the TIGER-compatible analyses, choose the most likely c-/f-structure pair according to the log-linear model for parse disambiguation.
4. Generate candidate(s) from the f-structure part of this analysis.
5. If the source string is contained in the set of output strings, add this sentence and all of its corresponding c-/f-structure pairs to the training set. The pair(s) that correspond(s) to the original corpus sentence was/were labeled as the preferred structure(s), while all others are marked as dispreferred.

In theory, when regenerating from a string, it is expected to have the original string amongst the alternative strings generated by the system. However, XLE often does not generate punctuation in all possible positions which explains the cases that the input string does not appear in the set of generated strings. Cahill et al. (2007b) didn't use such strings for training.

They evaluated their system using two metrics: exact match and BLEU score (Papineni et al., 2002). Exact match measures what percentage of the most probable strings are exactly identical to the reference sentence. BLEU score is a more relaxed metric and its major application is to evaluate Machine Translation (MT) systems. Applying it to text generation, it indicates the similarity of the highest ranked sentence with the reference sentence, by calculating the percentage of the n-grams from the former that also exists in the later, multiplied by a brevity factor. This factor is a ratio of the candidate sentence length to the reference sentence length.

Their results indicate that training on c-structure features alone is the worst in exact match and BLEU score, even lower than the baseline. They argue that this is caused by the nature of the employed features which were initially designed for parse disambiguation. Surprisingly, the log-linear model trained on language model features alone performs worse than the baseline language model applied directly; but still better than c-structure model. Looking into the models that combine two feature categories at a time—AF + LM, CF + LM and AF + CF—the first model achieved the greatest improvement in both evaluation metrics. The BLEU score was 0.7705, which is only a little less than the best result achieved by combining all three feature types (0.7808).

1. The groups who protested against plans to remove... (The GlasgowHerald; Jun 25, 2010)
2. .. to help cover small groups that can't get insurance ... (WSJ0518.)
3. Five miles is a long distance to walk. (Kim, 2004)
4. King prawns cooked in chili salt and pepper was very much better ... (Kim, 2004)

Figure 2.11: Agreement as a ranking problem.(Rajkumar and White, 2010)

They concluded that the language model features and the additional features had the most contribution to the model, while the c-structure features had the least. Nevertheless, the c-structure features were beneficial, since the best model was the one that combined the three feature types. Despite these encouraging results, an error analysis of the realisation ranking confirmed that further features are required to enhance the quality of the output strings.

Cahill et al. (2007b) additionally reported the quality of reranking to provide an indication of how close the correct string, the closest to the reference string, was to being selected. The authors calculated the ranking score as follows:

$$S = \begin{cases} \frac{n-r+1}{n} & \text{if } r \text{ is defined,} \\ 0 & \text{otherwise} \end{cases}$$

where n is the total number of generated alternative realisations and r is the rank of the gold standard string. If the gold standard string is not among the list of potentials, i.e. r is undefined, the ranking score is defined as 0. Since the original string was not necessarily in the set of candidate strings, they provided the upper bound, i.e. if the ranker had chosen the correct string any time the correct string was available. Their results indicated that in almost 62% of the cases, the original string was generated by the system. The ranking scores for hybrid model and language model alone were reported as 0.5437 and 0.4724, respectively. The ranking score once again proves the superiority of the hybrid model to the baseline by ranking the original string (when available) higher in the list of candidate strings.

2.2.1.3 Realisation Ranking using Log-linear with Combinatory Categorical Grammar (CCG)

So far, all the discussed works focused on ranking the generated text regarding the grammaticality of a sentence. Rajkumar and White (2010) tried to extract features that resolve both animacy agreement between nouns and relative clauses and number agreement for subj-verb agreement. Sentences 1 and 2 in Figure 2.11 are examples of the former agreement and sentences 3 and 4 are instances of the latter.

Traditionally, grammatical agreement has been handled using hard constraints in the grammar. The problem with hard constraints is they are not capable of handling the exceptions. Thus, the authors built a statistical realisation ranking model that learns animacy and number agreement to rank competing preferences. Competing preferences generally occur in text generation due to under-specification, e.g, alternatives with relative pronouns *who*, *that*, *which*, *etc*. They used a perceptron model⁹ for this purpose which also helped with reducing balanced punctuation errors so that a post-filter will be required to handle such cases. As they used Combinatory Categorical Grammar (CCG) formalism (Steedman, 2000), OpenCCG was their realiser and they trained and tested their system on an enhanced version of CCGbank (Hockenmaier and Steedman, 2007). They consider three different features for their perceptron model to solve animacy and number agreement:

1. a log probability of the generated realisation word sequences computed according to their linearly interpolated language model as previously suggested by Velldal and Oepen (2005);
2. syntactic features by implementing Clark and Curran (2007) normal form in OpenCCG; and
3. discriminative n-gram features.

For instance to rank sentences that contain relative pronouns, they engineered features so that they took into account the name entity class of the head word along with its stem. Consequently, the model learned that *who* should go with proper nouns (NNP and NNS) and *which* and *that* are appropriate for other Part of Speech (PoS) tags associated with nouns.

They were the first to prove that provided labelled data, linguistically motivated features can capture language-specific features such as agreement and animacy. They showed that incorporating such features in realisation ranking models could significantly improve the realisation ranking results in terms of both exact match and BLEU score.

2.3 Data-Driven Methods: Surface Realisation using Other Input

Back in 2010, Belz et al. reported that unlike PoS taggers and parsers, surface realisers had rarely been reusable. The authors attributed this to the wide variety of input representations used by various realisers and the lack of an established basis for comparing the surface realisers' output quality. As the first step to promote the reuse of surface realisers as a tool, the authors proposed a shared task in surface

⁹A kind of linear supervised classifier.

Team	Organisation(s)	Shallow systems	Deep systems
ATT	AT&T Labs Research	ATT-0 ^y	-
OSU	Ohio State University	-	OSU ^y
STUMABA	Universitat Stuttgart	STUMABA-S ^{x,y}	STUMABA-D ^{x,y}
	Universitat Pompeu Fabra Universite du Maine		
UCM	Universidad Complutense de Madrid	UCM	-

Table 2.4: SR-Task teams and systems.

^x = resubmitted after fixing software bugs;^y = late submission (Belz et al., 2011)

realisation. The main challenging aspect of their proposal was to define a common-ground representation formalism so that it can be consumed by various realisers through mapping to their individual representation (Belz et al., 2010).

They agreed that once the common input representations can be automatically derived from annotations in existing resources, then data can be produced in sufficient quantities to allow participants to automatically learn mappings from the system-independent input to their own input. They proposed a solution along the lines of what had been done in the CoNLL’08 shared task on Joint Parsing of Syntactic and Semantic Dependencies: combining the Penn Treebank, Propbank, Nombank and the BBN Named Entity corpus into a dependency representation. Their other challenge was to define an evaluation method.

The authors decide not to focus on single-best realisations, as it will not encourage research on producing all possible good realisations. To take into account the one-to-many nature of the realisation mapping, they proposed to adapt existing automatic intrinsic methods, e.g. BLEU or NIST, to calculate scores for the n-best realisations produced by a realiser and then to compute a weighted average where scores for realisations are weighted in inverse proportion to the ranks given to the realisations by the realiser. Also Clarity, Readability and Meaning Similarity can be assessed by human judges.

The first surface realisation task was one of the tasks in Generation Challenges 2011 (Belz et al., 2011) with five teams submitted six systems (see Table 2.4) in the following two strands: **shallow** mapping from shallow input representations to realisations and **deep** mapping from deep input representations to realisations. Both representations were sets of unordered labeled dependencies; see Table 2.5 for comparison of the two representations.

Belz et al. (2012) reported that the four top-performing systems (StuMaBaD, StuMaBa-S, DCU and ATT) were all statistical dependency realisers without an

Representation	Node	Edge
Shallow	<ul style="list-style-type: none"> ✓ word's lemma ✓ coarse-grained POS-tag ✓ number* ✓ tense and participle features* ✓ a sense tag id* punctuation features: <ul style="list-style-type: none"> ✓ quotation and bracketing ✓ comma ✓ function words (complementizers and TO infinitives) 	<ul style="list-style-type: none"> ✓ syntactic edges: Penn- nconverter syntactic labels ✗ semantic edges
Deep	<ul style="list-style-type: none"> ✓ word's lemma ✗ coarse-grained POS-tag where appropriate: ✓ number* ✓ tense and participle features* ✓ a sense tag id* punctuation features: <ul style="list-style-type: none"> ✓ quotation and bracketing ✗ comma ✗ function words (complementisers and TO infinitives) 	<ul style="list-style-type: none"> ✓ syntactic edges: Penn- nconverter syntactic labels ✓ semantic edges: Prop- bank and Nombank semantic roles la- bels.

Table 2.5: Comparison between Deep and Shallow representation.

* = where necessary

explicit, pre-existing grammar. The good result is attributed to the robustness and flexibility of statistical dependency realisers in consuming new kinds of dependency inputs. On the contrary, the only two systems that employed a grammar, either hand-crafted (UCM) or treebank-derived (OSU), did not produce competitive results. Both teams with grammar-based systems experienced substantial difficulties in mapping the common ground inputs into their systems' expected inputs.

A conclusion to be drawn is that realisers which do not depend on grammar-based systems are an avenue worth exploring. Below, we look at one class of approach that uses words and dependencies as inputs, which was inspired by graph-based dependency parsing.

2.3.1 Data-driven Dependency Parsing using Spanning Trees

Graph-based parsing uses a directed graph to represent all possible dependency relationships between words in a sentence, and the goal is to choose some subset that forms a good tree. Data-driven dependency parsing models, as the name implies, require annotated corpora to learn dependencies. There exist various models in the parsing literature but in this chapter, we only reference to those which inspired us to perform surface realisation tasks.

McDonald et al. (2005b) used an edge-factored model to provide a general framework for parsing trees for both projective and non-projective languages. Edge-factored dependency parsing model is a popular subclass of data-driven parsing that considers each edge as an independent factor in producing the final tree.

Figure 2.12 shows how Chu-Liu-Edmonds (CLE) — a standard algorithm in graph theory to locate Minimum Spanning Tree (MST) — can be applied to search space to extract an MST which serves as a dependency parse tree for a given sentence. In this figure, weights represent the likelihood of an edge as a dependency relation. In the absence of such algorithm, we can locate a couple of subgraphs that are non-maximal trees, e.g. the tree composed of the following edges: *root* to *John*, *John* to *saw* and *saw* to *Mary*. On the other hand, if we follow the greedy selection of edges to get the maximal subgraph, the outcome would not be necessarily satisfactory as it might not represent a tree, as in Figure 2.12(b). The CLE algorithm is given in Algorithm 1, and is discussed in more detail in its application to realisation in Section 2.3.2.

As suggested by the MST extraction process shown in Figure 2.12, the accuracy of the dependency tree depends on dependency weights. Therefore, McDonald et al. (2005a) proposed to adapt Margin Infused Relaxed Algorithm (MIRA) (Crammer et al., 2006) to train a model to generate dependency parse trees. Their adaptation resulted in a dependency parser with competitive accuracy without any word limit

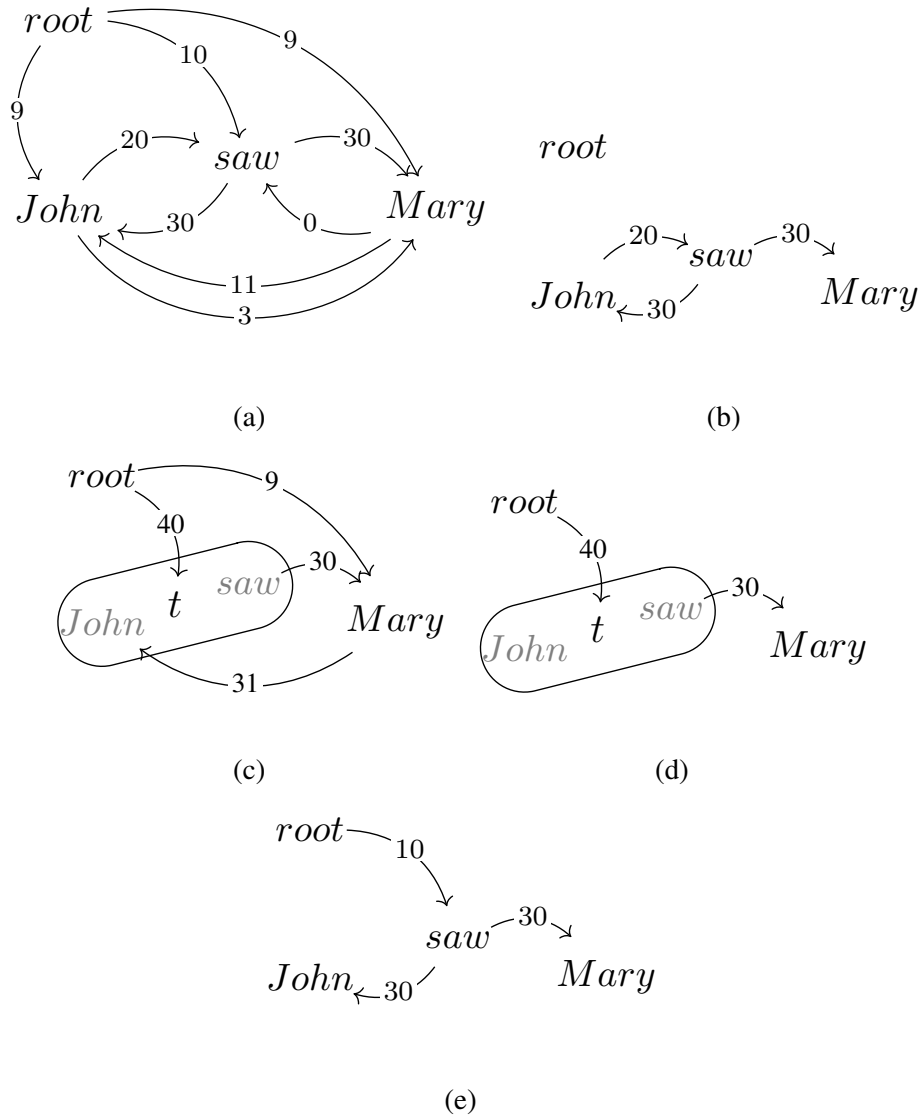


Figure 2.12: A step by step example of the CLE algorithm to get the MST for Sentence (5.1). (a) create initial digraph and assign weights to each dependency; (b) greedily choose incoming dependencies with minimum weight; (c) contract the cycle into a temporary node *t* and recalculate the weights; (d) once again, greedily choose incoming dependencies with minimum weight which results in a valid tree; (e) revert *t* to its original building nodes to get the MST (McDonald et al., 2005b).

on the sentence, as opposed to Taskar et al. (2004)’s attempt to formulate parsing with max-margin which had a word limit of 15.

McDonald and Satta (2007) provided an extensive comparison between projective and non-projective parsing in addition to an indepth study of the problematic key areas in learning and inference methodologies for graph-based dependency parsing: (1) Finding the Argmax, (2) Calculating the partition function and (3) Assigning edge weights, by looking into existing algorithms. The authors showed that each of the above mentioned values can be tractably calculated for the non-projective in polynomial time with the assumption that dependency relations are mutually independent.

The authors also discussed the limitations of edge-factored such as horizontal/vertical Markovization and incorporation of arity. The above mentioned independency assumption does not allow a given edge to look at its neighbourhood, i.e, lack of non-local information which results in trees of unbounded arity, and neighbouring dependencies which can be crucial to obtaining high parsing accuracies.

As a result, exhaustive methods have been proposed to weaken edge-factored assumptions including both approximate methods (McDonald and Pereira, 2006), and exact methods through Integer Linear Programming (ILP) (Riedel and Clarke, 2006) or branch-and-bound algorithms (Hirakawa, 2006).

2.3.2 Parse Trees as Intermediate Structures for String Regeneration

Wan et al. As noted at the start of Section 2.3, there’s a range of generic input representations that aren’t tied to particular grammar formalisms, which would be particularly applicable in cases where grammar resources aren’t available at all. Wan et al. (2009) considered the case of the input being just a bag of words, and had as their goal (similar to the reranking work discussed in Section 1.2) to regenerate the original sentence. The idea in that work was to induce some partial structure to assist the word ordering. In particular, they drew on the ideas of McDonald et al. (2005b) to assign an MST to a bag-of-words to serve as a guiding structure. They addressed the problem by redefining it as a graph-theory problem: finding the Minimum Spanning Tree (MST) to serve as a dependency tree. Wan et al. (2009) mapped the search space into a complete weighted digraph, so that each node represents a word and each edge denotes a potential dependency. Figure 2.13 represents such a digraph for a bag-of-words consisting of n nodes, where N_i , $i \in 1..n$, is the i th word, and w_{ij} represents the weight of the dependency where N_i is the head and N_j is the dependent.

Wan et al. adapted Collins (1996)’s definition to assign edge weights using Maximum Likelihood Estimate (MLE). For a given edge $e = (u, v) \in E$ and

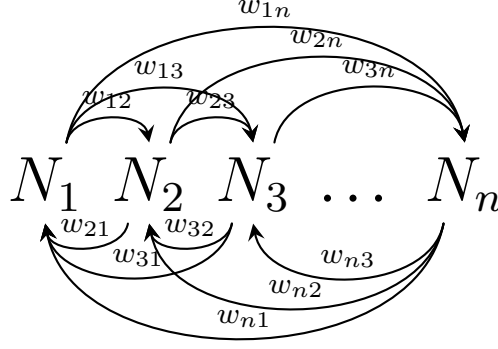


Figure 2.13: A complete digraph for a bag-of-words of size n . Each N_i represents a word and each w_{ij} denotes the weight for a dependency which N_i is the head and N_j is the dependent .

direction $d \in \mathcal{D}$ consulting the Penn Treebank, the edge weight is calculated as:

$$s((u, v), d) = -\log \text{prob}_{dep}(u, v, d) \quad (2.5)$$

$$\text{prob}_{dep}(u, v, d) = \frac{\text{cnt}((u, \text{pos}(u)), (v, \text{pos}(v)), d)}{\text{co-occurs}((u, \text{pos}(u)), (v, \text{pos}(v)))} \quad (2.6)$$

where $s((u, v), d)$ is the log probability of the number of the times that $(v, \text{pos}(v))$ is modifying $(u, \text{pos}(u))$ in direction d divided by the number of the times $(u, \text{pos}(u))$ and $(v, \text{pos}(v))$ co-occur in a sentence in the training set. Following Collins (1996)’s work, they back off to PoS, for unseen dependency samples. They used the CLE algorithm (see Algorithm 1), as McDonald did for parsing.

Figure 2.12 shows how CLE is applied to sentence (5.1) step-by-step to produce an MST for it. For each node, this algorithm picks the incoming edge with minimum weight (Figure 2.12(b)). If these edges altogether form a tree then the task is done, otherwise there must be at least one cycle. The graph is then modified so that each cycle (C) is contracted to one node and the weight for the outgoing edge to node a , where $a \notin C$, would be equal to the highest scoring edge from any vertex of the cycle-participating nodes. The incoming edge weight from a given node a , where $a \notin C$, would be the score of the best spanning tree originating from a that includes the vertices in the cycle (Figure 2.12(c)). The algorithm will repeat choosing edges with minimum weights and applying contraction steps until no cycle exists in the graph (Figure 2.12(d)). Finally, all the contracted node(s) must be expanded back to their original state to get the final MST (Figure 2.12(e)).

Since CLE is a pure MST algorithm, it fails to incorporate linguistic properties: for example, the tree’s branching factor might not be appropriate for verb valency.

Algorithm 1 Chu-Liu-Edmonds Algorithm with adaption to include linear precedence (Wan et al., 2009)

```

1   $T_w \leftarrow (u, v) \in E : \forall_{v \in V, d \in D} \arg \min_{(u, v)} s((u, v), d)$ 
2  if  $M_w = (V_w, T_w)$  has no cycles then
3    return  $M_w$ 
4  end if
5  for all  $C \subset T_w : C$  is a cycle in  $M_w$  do
6     $(e, d) \leftarrow \arg \min_{e^*, d^*} (se^*, d^*) : e \in C$ 
7    for all  $c = (v_h, v_m) \in C$  and  $d_c \in D$  do
8      for all  $e' = (v_i, v_m) \in E$  and  $d' \in D$  do
9         $s(e', d') \leftarrow s(e', d') - s(c, d_c) - s(e, d)$ 
10     end for
11     end for  $s(e, d) \leftarrow s(e, d) + 1$ 
12  end for
13   $T_w \leftarrow (u, v) \in E : \forall_{v \in V, d \in D} \arg \min_{(u, v)} s((u, v), d)$ 
14  return  $M_w$ 
```

To tackle this problem, Wan et al. (2009) introduced an Argument Satisfaction Model termed the Assignment-Based (AB) system. In this model they defined a maximum number of argument positions for different word categories, e.g. nouns and verbs. Hence for a given head v , function q maps it into a set of attachment positions $\{v_{dj}\}$, where d specifies whether the modifier is a right branch attachment (r) or left one (l), and j states the relative order on each side:

$$q(v) = \begin{cases} \{v_{r1}\}, & \text{if } v = w_0, \text{ the root} \\ \{v_{ri}, v_{lj} | i, j \in \mathbb{N}\}, & \text{if } \text{pos}(v) \text{ is a verb and } v \text{ is the child of } w_0 \\ \{v_{lj} | j \in \mathbb{N}\}, & \text{otherwise} \end{cases}$$

q can be defined with more sophistication, for example by constraining based on PoS tags in order to strengthen the linguistic aspect of dependency selection.

Wan et al. also allow a two-way competition: head words' argument positions compete with each other to find a modifier, and at the same time, modifiers compete with each other to be assigned to a head. To obtain a balanced dependency tree a branching factor, the maximum number of dependents per head, is specified for each node. Sentence (2.3) and Figure 2.14 illustrate a tree where such constraints have been applied to e.g. the verb *remains*, to enforce a typical valency; similarly for the preposition *in*. Once an edge is assigned to a node which has already gone past its maximum branching factor, a penalty is applied to the weight of that edge.

(2.3) Concerning your Sept. 21 page-one article on Prince Charles and the

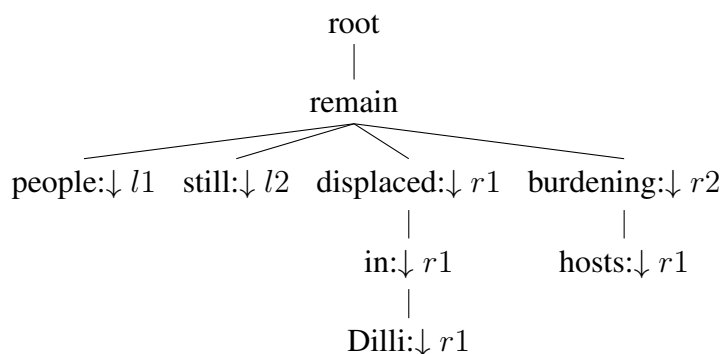


Figure 2.14: The spanning tree of the sentence: *People still remain displaced in Dili, burdening hosts*. Pages 188–189 of Wan et al. (2009) describe the process step-by-step.

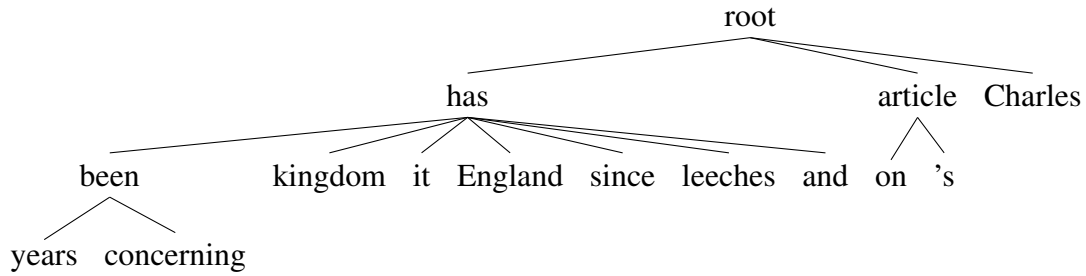
leeches : It 's a few hundred years since England has been a kingdom.

This dynamic weight change of the graph makes the problem an NP-hard problem (McDonald and Satta, 2007). They represent the problem of finding the dependency tree as an instance of the assignment problem. The assignment problem is an optimisation problem and consists of finding a maximum weight matching (or minimum weight perfect matching) in a weighted bipartite graph where a match in a graph is a set of edges without common vertices. Consequently, instead of CLE, they applied the Hungarian algorithm which is one the standard algorithms that solve this problem in a polynomial time. Figure 2.15 compares two trees produced by the CLE and AB algorithms. The final word sequence is determined by traversing the tree using a greedy edge selection algorithm. However, it is a crucial task to decide the order of the modifiers of an individual head with the same dependency direction; to do this, an n-gram language model score is calculated for all possible word orders.

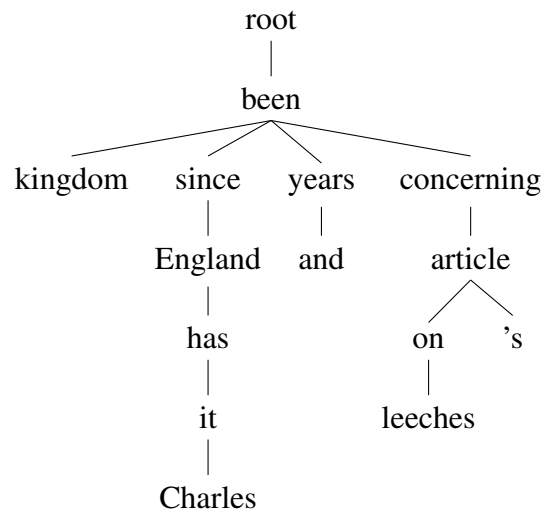
Other works Similar to Wan et al. (2009), Zhang and Clark (2011) addressed the word-ordering problem by finding the best parse tree in the search space. However, instead of inducing structure via a graph-based dependency parse method, they use CCG parsing as described below.

They extracted the grammar by reading rule instances from CCGbank¹⁰ (Hockenmaier and Steedman, 2007) to guide the parse tree creation. This grammar is used to construct the parse trees in a bottom-up fashion, starting from single words. The algorithm uses beam search to locate the highest score hypothesis and expands it. The expansion was performed by either applying unary rules to a hypothesis or

¹⁰A translation of the Penn Treebank into a corpus of CCG derivations.



(a) MST for sentence (2.3) using CLE algorithm.



(b) MST for sentence (2.3) using Hungarian algorithm.

Figure 2.15: Comparison of CLE and Hungarian algorithms output. Comparing trees in 2.15(a) and 2.15(b) shows the effectiveness of using branching factor in the later.

binary rules to combine existing hypotheses. The complexity of each hypothesis expansion is kept linear in the size of a beam.

Later work by Zhang et al. (2012) showed that adding n -gram language model scores to such a system improves the output. The authors redefined the weight for edge e to be linear interpolation of syntax model $f(e)$ and n -gram score $g(e)$ as:

$$\begin{aligned} F(e) &= f(e) + g(e) \\ &= f(e) + \alpha \cdot g_{four}(e) + \beta \cdot g_{tri}(e) + \gamma \cdot g_{bi}(e) \end{aligned} \quad (2.7)$$

where g_{four}, g_{tri} , and g_{bi} are the log probabilities of the corresponding n -gram language model score and α, β , and γ are score weights.

In the latest and best performing model, Zhang (2013) switched to dependency features for the search algorithm, sticking to the same training framework as Zhang and Clark (2011). This work demonstrated that BLEU score improvement in Zhang et al. (2012) was due to the guided search framework, not the features' type.

Linguistic constraints are hardcoded in both approaches proposed in Wan et al. (2009) and Zhang and Clark (2011). However, there are many possible linguistic constraints, so we want to be able to add arbitrarily many: as an example consider the tight coupling between the language and the domain. The former approach allows an easier translation to a declarative approach compared to the latter's bottom-up approach. Therefore, we take the approach of Wan et al. (2009) as our starting point to define a declarative string regeneration framework.

It is worthwhile to note that the correspondence between an unordered dependency tree and a sentence has been also valued by researchers with a more linguistic focus. Kahane and Lareau (2016) showed that linearisation can be seen as a graph rewriting process. However, unlike what we pursue, less reliance on crafted grammar, they took a mathematical/theoretical approach — not computational — and proposed a general formalism for a dependency grammar that can be used for both parsing (sentence to dependency tree) and realisation (dependency tree to sentence).

2.4 Creative NLG Systems

The immense rise of processing power combined with new techniques and architecture that addressed the pitfalls of traditional Neural Network (NN) — such as getting stuck in local optimas and training deep networks — contributed to the successful re-emergence of these methods under the name Deep Learning for solving various problems across fields of Artificial Intelligence (AI) such as image recognition and speech processing. The recent improvements to Neural Net-

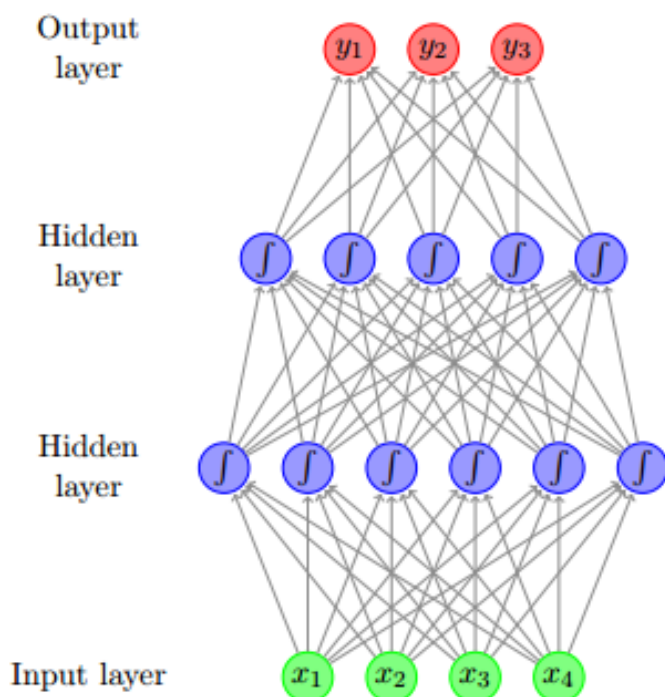


Figure 2.16: feed-forward neuralnetwork with two hidden layers, reproduced from Figure 2 from Goldberg (2015).

work (NN) enabled them to model the NLP problems and increase their popularity. Goldberg (2015) provided a substantial tutorial survey on NNs for NLP; thus in this section, we briefly recap two popular NN architectures and how they deal with textual input.

NNs are a loose inspiration of computational model that distributes the process to a multi-layer network of neurons. Figure 2.16 depicts a feed forward NN. The very first layer feeds the input to the network and the top most layer returns the output. The layers in between are called hidden layers and traditionally, each node in layer _{i} is connected to all the nodes in layer _{$i+1$} using a weight. This weight indicates the importance of the given connection. Each neuron applies a non-linear function, such as the sigmoid function, and passes it to the next layer. Such networks are called feed forward for the forward flow of data. These traditional networks are suitable for sequence-independent tasks with fixed-size vector as input/output. Sequence-dependent tasks with variable size input/output are hard to be modelled by plain NNs, since the variable-size sequence must be transformed, if possible at all, into a fixed-size input. As a consequence, some structural properties of the sequences such as order might be discarded which is a significant heuristic for language based problems. Another limitation of NNs is the fixed amount of

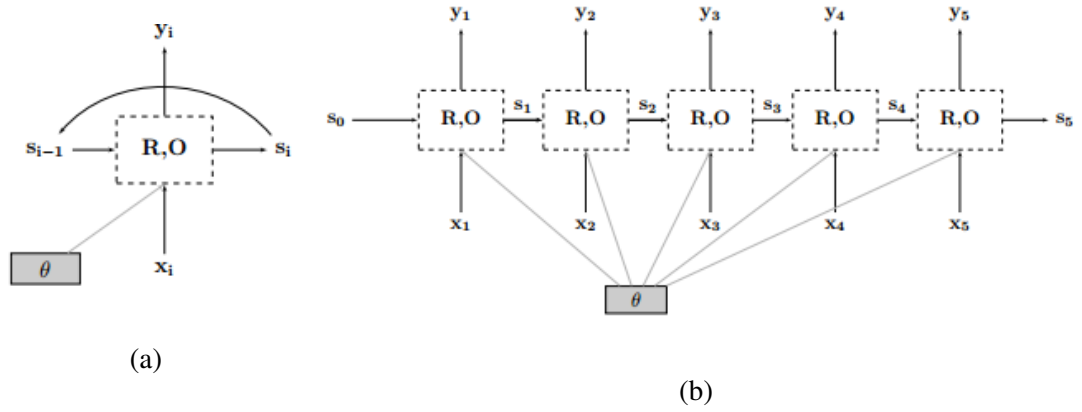


Figure 2.17: 2.17(a) the unrolled graphical representation of an RNN, reproduced from Figure 5 from Goldberg (2015). 2.17(b) The rolled graphical representation of an RN, reproduced from Figure 6 from Goldberg (2015).

computational steps, e.g., layers.

Recurrent Neural Network (RNN)s are a variation of NN that address the above mentioned shortcomings of NNs by introducing a flexible sequence-based computational model. Each neuron in an RRN applies function R to its input x_i to produces state s_i . R is a recursive function that relies on the previous neuron's state s_{i-1} , for instance s_4 is calculated as follows:

$$\begin{aligned}
 s_4 &= R(s_3, x_4) \\
 &= R(R(s_2, x_3), x_4) \\
 &= R(R(R(s_1, x_2), x_3), x_4) \\
 &= R(R(R(R(s_0, x_1), x_2), x_3), x_4)
 \end{aligned}$$

This recursion is the characteristic of RNNs that incorporates sequence for more effective modelling. In addition to R , each neuron maps its state s_i into output y_i using a function O . Figures 2.17(a) and 2.17(b) present the recursive and unrolled graphical representation of an RNN, respectively. The former representation is suitable for arbitrarily long sequences whilst the latter is the unrolled version of recursive representation and best describes a finite size input sequence. It is noteworthy that the R and O functions are the same across the network. Also the parameter θ that is included in the representation indicates that the same parameters are shared across all the states. The example in Figure 2.18 shows a few examples of RNNs handling sequential input/output or both.

It has only been recently that RNNs have been deployed for text generation (Wen et al., 2015, Sutskever et al., 2011). Provided a large corpus for training, these networks are capable of generating at character level, i.e. they are able to produce a text by generating one character at a time. Lately, Andrej Karpathy publicly shared a code for such an RNN¹¹. This means they require no lexicon, no document plan, no content selection, and no grammar. However, as briefly mentioned in the Chapter 1, texts produced with such systems lack coherent semantics. What they essentially do, at the time of writing this thesis, is more sequence modelling, and less generating text with the purpose of communicating with the human reader.

Given the current approach of using deep learning methods at this stage, such systems can be beneficial to tasks with no context, like image captioning. For instance, Karpathy and Li (2015) introduced an image captioning system¹² that is trained over a set of images and their corresponding sentence descriptions. They combined a Convolutional Neural Networks over image regions, bidirectional Recurrent Neural Networks over sentences, and a structured objective that aligns the two modalities through a multimodal embedding.

Manning (2015), who acknowledges the success of deep learnings methods in producing state-of-the-art results, used a tsunami as the analogy for the recent popularity of deep learning in major NLP publications. He argued that such methods would not solve NLP, mainly due to the domain specific nature of language. He attributed the gains reported for NLP problems to the use of distributed word representations than from the deep learning itself — the use of a hierarchy of more abstract representations to promote generalisation. The author encouraged researchers to return some emphasis within NLP to cognitive and scientific investigation of language rather than almost exclusively using an engineering model of research that aims to beat numbers. He suggested the field requires more efforts into problems, approaches and architectures, for instance if meaning composition functions can be built into Deep Learning systems.

In this thesis, we embrace his idea and focus on the approaches to text generation problems with regards to minority languages within the consensus architecture noted above. Given that existing deep learning approaches have similar problems in terms of grammaticality of output, it is possible that some of the ideas we propose — for example, the imposition of soft grammaticality constraints on output — could be adapted for those.

¹¹<https://github.com/karpathy/char-rnn>

¹²The source code is available at <https://github.com/karpathy/neuraltalk2>

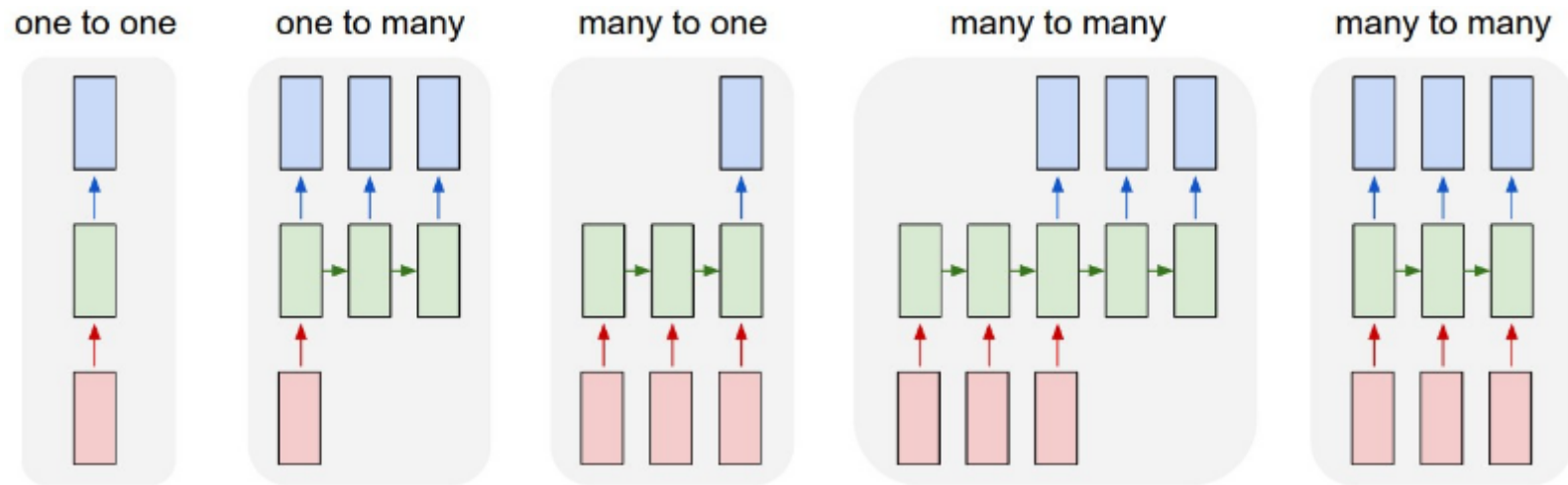


Figure 2.18: Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: (1) Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). (2) Sequence output (e.g. image captioning takes an image and outputs a sentence of words). (3) Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). (4) Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). (5) Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case there are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like (reproduced from an example provided in <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>)

2.5 Summary

In this chapter, we reviewed the existing literature of grammar-based realisation ranking and realisation from bag-of-words input. Our aim is to identify resources and methods to address these two tasks with regards to languages that lack computational resources: for example, have a grammar still being developed that could benefit from cheaply obtained features, or have no grammar at all. In the case of grammar-based realisation ranking, we focussed on the log-linear models of e.g. Cahill et al. (2007b), which allow arbitrary features to be included. In Chapters 3 and 4 we go on to investigate features for a log-linear model derived from parsers that could, for a language lacking computational resources, usefully complement a grammar.

We also reviewed generation from a bag of words, with a focus on the model of Wan et al. (2009), which adapted graph-based parsing methods to realisation. A key insight in that work was to construct trees during the realisation process that incorporated linguistic properties. In chapters 5 and 6 we focus on regeneration from a bag-of-words. We combine the strength of the ILP methods incorporation of declarative constraints with the learning capability of MIRA in learning dependencies between words, previously proved to be useful in dependency parsing.

Chapter 3

Grammatical Structures for Reranking

3.1 Introduction

In this chapter, we approach the realisation reranking problem, previously discussed in Section 2.2, from the low-density language perspective¹ defined in pageref ch1:lowdensitydef. To recap, all rankers depend on multiple structures produced by the respective large-scale symbolic grammars to rank the output. For much smaller symbolic grammars, and those in the process of development — e.g. Lexical Functional Grammar (LFG) grammars for Indonesian (Arka et al., 2009) or Arrernte (Dras et al., 2012), or Head-driven Phrase Structure Grammar (HPSG) grammars for Persian (Müller, 2010) or Wambaya (Bender, 2008) — these multiple structures will not be available or will be quite impoverished. The supervised approach is an ideal upper bound from a resource-rich language, and the techniques that are presented in this chapter aim to close the gap by looking into alternative resources for the extraction of the structural features.

We investigate multiple reranking models using features from various statistical parsers as potential alternatives to manually-crafted grammars. We take n-gram language model (LM) score, the traditional resource for reranking, as our baseline and start off by deploying a set of templates from a supervised parser as features, and do the same with an unsupervised parser. As we anticipate, the unsupervised parser features would lead to lower quality features, we also study the effect of two feature selection methods to close the gap between the resource-rich and resource-poorer languages.

In Section 3.2, we describe work we will draw on in this chapter, in particular work

¹The work presented in this chapter was published in Motazedi et al. (2012).

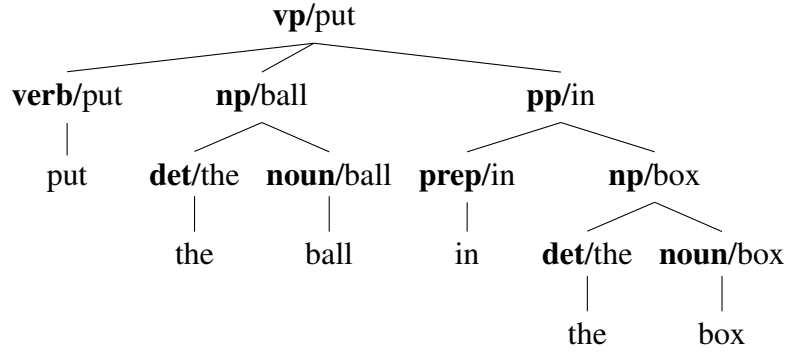


Figure 3.1: A tree showing head information (Charniak, 2001).

on parser-based models for ranking and on unsupervised parsing. In Section 3.3, we describe our models based on statistical parsers, and the experimental set-up for investigating them. In Section 3.4, we discuss our results, and conclude in Section 3.5.

3.2 Related Work

We reviewed the use of ranking in the realisation in Section 2.2. The other work we draw on in this chapter is the use of statistical parsers in ranking, and work on unsupervised parsing; we review these below.

3.2.1 Parser-based Ranking

Charniak (2001) initiated the use of immediate-head parser as a source for structural language model. They implemented a *generative immediate-head* parser:

immediate-head means all of the properties of the immediate descendants of a constituent c are assigned probabilities that are conditioned on the lexical head of c . As an example in Figure 3.1 the probability that the **vp** expands into **v np pp** is conditioned on the head of the **vp**, "put". Formulated precisely in (3.1), the probability of each parse tree $p(\pi)$ is calculated in a top down manner so that for each constituent c , first guessing the Part of Speech (PoS) tag of c , $t(c)$ (t for "tag"), then the lexical head of c , $h(c)$ (label of c , such as

np), and finally the expansion of c into further constituents $e(c)$:

$$p(\pi) = \prod_{c \in \pi} p(t(c)|l(c), H(c)) \quad (3.1)$$

$$\cdot p(h(c)|t(c), l(c), H(c))$$

$$\cdot p(e(c)|l(c), t(c), h(c), H(c))$$

where $H(c)$ is the relevant history of c captured from the context outside of c , such as label, head and PoS tag for the parent of c .

generative means for a sentence s , the parser tries to find the parse π defined by :

$$\arg \max_{\pi} p(\pi|s) = \arg \max_{\pi} p(\pi, s)$$

By calculating $p(s) = \sum_{\pi} p(\pi, s)$ each sentence can be assigned a probability which then can be used for ranking sentences, in a manner analogous to n-gram language models. That is the structural language model.

The idea has been applied a number of times in Machine Translation (MT), for example by Charniak et al. (2003) or Post and Gildea (2008); it has also been applied in Natural Language Generation (NLG), where Mutton et al. (2007) showed that a combination of parser-based metrics correlated with human judgements of the quality of generated text. To our knowledge, all work applying parser-based ranking has used supervised parsers. There is also an interesting piece of work by Cherry and Quirk (2008) where implicit discriminative syntactic LMs are constructed, using a latent Support Vector Machine (SVM) to train an unlexicalised parser to judge sentences produced by an MT system. The authors discuss some similarities to unsupervised parsing, in that both sorts of parsers are trained on sentences without the benefit of annotated parse trees. Using unsupervised parse trees as we do in this chapter, however, can benefit from lexicalisation and, potentially, linguistic knowledge embodied in the parser (see below).

3.2.2 Unsupervised Parsers

Unsupervised parsing refers to the induction of a statistical parsing model from raw text. The first unsupervised parser that convincingly beat fairly simple baselines, such as constructing a right-branching tree, was the Dependency Model with Valence (DMV) model of Klein and Manning (2004). In their model, a dependency (sub)tree $T(h)$ with root (h) is generated as follows:

$$P(T(h)) = \prod_{d \in l, r} \left[\prod_{a \in D(h, d)} P_l(\neg! | h, d, ?) P_v(a | h, d) P(T(a)) \right] P_l(! | h, d, ?) \quad (3.2)$$

For each observed coarse symbol s :

1. Draw top-level infinite multinomial over subsymbols $\beta_s \sim \text{GEM}(\gamma)$.
2. For each subsymbol z of symbol s :
 - (a) Draw word emission multinomial $\phi_{sz} \sim \text{Dir}(\phi_0)$.
 - (b) For each context value c :
 - (i) Draw child symbol generation multinomial $\theta_{szc} \sim \text{Dir}(\theta_0)$.
 - (ii) For each child symbol s' :
 - (A) Draw second-level infinite multinomial over subsymbols $\pi_{s'szc} \sim \text{DP}(\alpha, \beta_{s'})$.

foreach tree node i generated in context c by parent symbol s' and parent subsymbol z' :

1. Draw coarse symbol $s_i \sim \text{Mult}(\theta_{s'z'})$.
2. Draw subsymbol $z_i \sim \text{Mult}(\pi_{s's'z'c})$.
3. Draw word $x_i \sim \text{Mult}(\phi_{s_iz_i})$.

Table 3.1: The generative process for model parameters and parses, where s is an observed coarse symbol, z is a hidden refined subsymbol, and x is an observed word. s' and z' represent the parent of the current node symbol and subsymbol. In the above GEM, DP, Dir and Mult refer respectively to the stick breaking distribution, Dirichlet process, Dirichlet distribution, and multinomial distribution. Reproduced from Table 2 in Naseem et al. (2010).

where d is the direction of the dependency — l for left and r for right. $D(h, d)$ is the set of dependents of h in direction d ; $P_l(h, d, ?)$ is the probability of stopping the generation of dependents in direction d and $?$ is a binary variable indicating whether any dependents have been generated or not; $P_v(a|h, d)$ is the probability of generating the dependent word a , conditioned on the head h and direction d and $P(T(a))$ is (recursively) the probability of the subtree rooted at a .

Bod (2007) combined DMV with Constituent Context Model (CCM) model and reported improvements for not only unsupervised parsing but also phrase structure parsing and dependency parsing. There have been various developments in unsupervised parsing since then: Headden III et al. (2009) introduced basic valence frames and lexical information, along with a smoothing technique to handle resulting data sparsity; Berg-Kirkpatrick and Klein used a phylogeny-structured model of parameter drift; and Naseem et al. (2010) based their work on the hypothesis that universal linguistics knowledge can improve unsupervised dependency grammar induction. Their parser consisted of (1) a probabilistic model that explains how sentences are generated from latent dependency structures and (2) a technique for incorporating declarative constraints into the inference process.

Root \rightarrow Verb	Verb \rightarrow Noun	Noun \rightarrow Adjective
Root \rightarrow Auxiliary	Verb \rightarrow Pronoun	Noun \rightarrow Article
Auxiliary \rightarrow Verb	Verb \rightarrow Adverb	Noun \rightarrow Noun
Preposition \rightarrow Noun	Verb \rightarrow Verb	Noun \rightarrow Numeral
Adjective \rightarrow Adverb		

Table 3.2: The manually-specified universal dependency rules used by Naseem et al. (2010)

Probabilistic model takes as input a set of sentences where each word has been tagged with a coarse PoS tag² (see Table 3.1 for more technical detail). Similar to the DMV model, after a node is drawn, children are generated on each side until a designated STOP symbol is produced. The authors encoded more detailed valence information than the original DMV model (Klein and Manning, 2004) and condition child generation on parent valence. The authors distinguished their work from the original DMV model in the following aspects: (1) it explicitly generates words x rather than only part-of-speech tags w , (2) it encodes richer context and valence information, and (3) it imposes a Dirichlet prior on the symbol distribution θ .

Inference with constraints The authors constrained the posterior to satisfy the rules in expectation during inference. This effectively biases the inference toward linguistically plausible settings. For this purpose the authors compiled a set of language-independent rules over coarse PoS tags (see Table 3.2).

Naseem et al. (2010) concluded that encoding a compact, well-accepted set of language-independent constraints significantly improves accuracy on languages like English, Danish, Portuguese, Slovene, Spanish, Swedish.

We use this last one in our work, as it gives state-of-the-art results as of 2010, and embodies potentially useful hard-coded universal tendencies. Notwithstanding that state-of-the-art performance, directed dependency results for that system range from only 50.9% (Slovene) to 71.9% (English), and importantly, as in previous work, these cross-linguistic evaluations are only carried out on sentences of 10 or fewer words.

Since the work described in this chapter was published, transfer parsing has arisen as an alternative to unsupervised parsing. We will discuss this type parser later in the end of this chapter.

²For instance, the coarse-grained PoS tag **Noun** covers for fine-grained PoS tags like **NN**, **NNS**, **NNP**, **NNPS**, **NP**.

3.3 Experimental Setup

Our goal is to evaluate the usefulness of the structural information proposed by an unsupervised parser in assessing the quality of sentences generated by a symbolic grammar, given its noticeably lower quality than supervised parsers. Like Cahill et al. (2007b) (described in Section 2.2.1.2, we use the Xerox Linguistics Environment (XLE) system and construct a symmetric treebank by parsing and re-generating sentences: this gives both positive examples (those that match the original sentence) and negative examples (those that don't). Details of the various aspects of this process follow.

3.3.1 Data and Evaluation

We took the Penn Treebank sections 2–21 for training, consisting of 38008 sentences, and section 23 for testing, consisting of 2245 sentences. We parsed them using XLE and the large-scale ParGram grammar of English (Butt et al., 2002), and then used XLE's re-generate facility to produce the multiple candidate realisations. As Cahill et al. (2007b) did, we found that XLE could not parse and re-generate all sentences; and for the purposes of constructing a training set, only instances that re-generated more than one sentence were useful³. In addition, we excluded the few sentences that contained more than 1000 re-generated sentences. This resulted in a training set of 20613 (sets of) sentences and a test set of 1168 (sets of) sentences.

Also as Cahill et al. (2007b) did, we found that not all sets of re-generated sentences contained the original sentence in its exact form: this was often because of small differences such as use of abbreviations (e.g. *Nov* for *November*) or use of punctuation. In contrast to their approach, where they discarded these cases, we used minimum edit distance (MED) to determine the re-generated candidate closest to the original.

Following this, we constructed training and test sets of pairs of re-generated sentences by taking as the positive example the one with smallest MED from the original sentence, and as the negative example one of two alternatives: either the one with largest MED (**greatest**) or a random selection from those candidates that did not have the smallest MED (**random**). **greatest** always gave the higher result in the classification experiments below, often by several percentage points (which is not surprising, as the differences are larger and the classification hence easier), so we mostly only report results for **random**. We used the maximum entropy learner

³There is also a technical issue in re-generating numerals. To get around this problem, we preprocessed the text to change numerals to those that could be re-generated.

MegaM (fifth release) by Hal Daumé III.⁴ Our evaluation metric is the classification accuracy of predicting the positive example from each pair. We took a slightly different approach by opting for binary classification rather than assigning a score to each alternative as Cahill et al. (2007b) did. In this thesis, ranking model is built upon pairwise evaluation which proved to be useful in other NLP applications such as MT, see Dras (2015). As discussed in this research, this method is a common approach for human ranking of the goodness of the sentences. In this thesis we report on ranking results over just a pair out of many possible alternatives for a given sentence. In a real-world scenario this process can be altered so that all the alternatives get ranked when the pairwise ranking is applied repeatedly.

3.3.2 Parsers and Language Models

Before answering any questions about the usefulness of unsupervised parsers, we address the question, Do *supervised* parsers produce structural information that is useful for realisation ranking? If the answer is yes (and based on the work cited in Section 3.2 and others, we would think it likely), the supervised parsers and a large LM would constitute an upper bound on the efficacy of using parsers to rank candidate sentences generated by XLE. The supervised parsers we use are the Stanford parser (Klein and Manning, 2003) and the Charniak and Johnson (henceforth C&J) parser (Charniak and Johnson, 2005). These parsers are both quite accurate: the Stanford parser gets a labelled f-score of 85.61 on the WSJ, and the C&J 91.09.

From the Stanford parser we examined both horizontal slices of parse trees, in effect treating them as sets of CFG production rules, and dependencies; the production rules and dependencies were either lexicalised or unlexicalised, and the dependency relations either named or unnamed. This gives two constituency and four dependency feature representations from the Stanford parser: *prod-rule-lex* and *prod-rule-unlex*; and *dep-lex-named*, *dep-lex-unnamed*, *dep-unlex-named* and *dep-unlex-unnamed*.

C&J is a reranking parser; the reranker uses 13 feature schemas such as tuples covering head-to-head dependencies, preterminals together with their closest maximal projection ancestors, and subtrees rooted in the least common ancestor. We took two types of features from C&J: production rules; and the instantiated feature schemas from the parse reranking process, making them do ‘double duty’ as realisation ranking features. C&J is used as a complement to the Stanford parser here, with respect to production rules, as an easy way of getting something approximating the compound features used in realisation ranking discussed in Section 3.2.

⁴MegaM software is available on <http://www.cs.utah.edu/~hal/megam/>.

The large LM was constructed using SRILM (Stolcke, 2002) on the Gigaword corpus.⁵ We use the 64K 4-gram and the 64K 2-gram (where the size nK represents the use of the top n words occurring in the training text as vocabularies), which are at the two extremes of size. We refer to these as l-lm4 and l-lm2.

For the unsupervised structure that is the main interest of the chapter, we used the parser of Naseem et al. (2010),⁶ which constructs dependency trees. Relation names are not inferred, so the two alternative representations are lexicalised (unsuper-lex) and unlexicalised (unsuper-unlex).

For a small LM that would be of a realistic size for the scenario we are interested in — i.e. the development of a symbolic grammar for a language with few resources — we considered the size of corpora produced by *An Crúbadán*⁷ (Scannell, 2007). This is a web crawler whose specific goal is the “automatic development of large text corpora for minority languages”. As examples, it has produced corpora of ~2M words for Akan (Ghana), ~5M words for Tamil, and ~7M words for Turkmen. Consequently, we use the Penn Treebank (~4M words) as our realistically-sized LM. On this sized corpus, we derive both 4-gram and 2-gram LMs from SRILM using the default settings; we refer to these as s-lm4 and s-lm2.⁸

3.3.3 Models

3.3.3.1 Base Models, Supervised and Unsupervised

The basic models are then the structural models described above. We used the LMs both as baselines and in combination with the structural models. (Note that the C&J parser already includes a LM in its reranking feature templates that we use, so we do not combine in this case.) In the base cases, we use all structures (production rules, templates or dependencies as appropriate) returned by the relevant parser.⁹

⁵<http://www.keithv.com/software/giga/> was the source for these models. They used interpolated, modified Kneser-Ney smoothing, bigram cutoff 3, trigram cutoff 5.

⁶<http://groups.csail.mit.edu/rbg/code/dependency/>

⁷<http://borel.slu.edu/crubadan/>

⁸We note that these are in-domain LMs, in contrast to the Gigaword-derived ones, and so will have a bit of an advantage.

⁹We treated positive and negative instances with identical feature representations differently when building the training set and test set. In the former we discarded the identical pair as they provide no information for training, and in the test set we kept the instances and made a random choice for assigning a class to each.

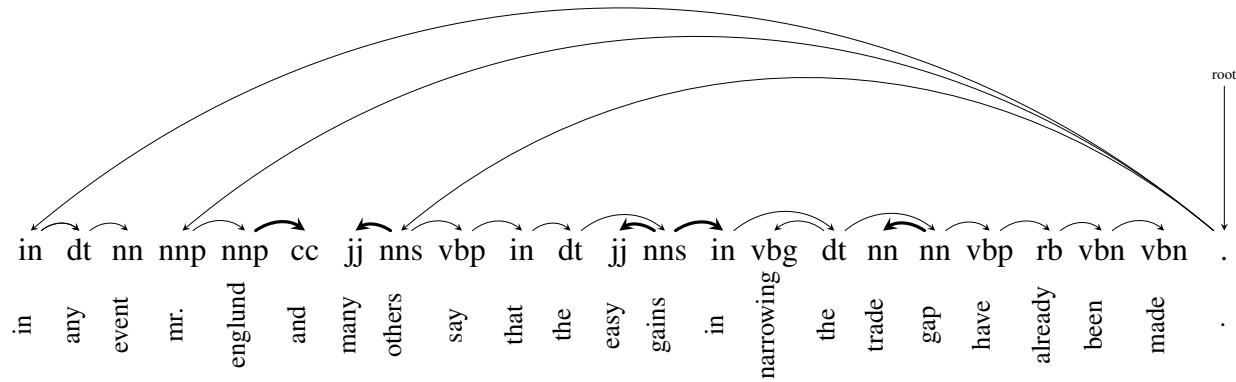


Figure 3.2: Unsupervised dependency tree for a sample sentence. Edges in bold are the dependencies that are identified by both supervised and unsupervised parser.

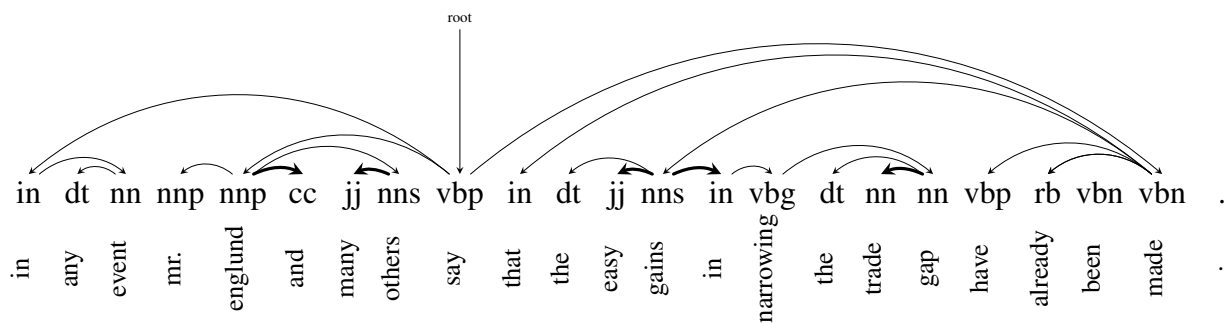


Figure 3.3: Supervised dependency tree for sample sentence of Fig 3.2. Edges in bold are the dependencies that are identified by both supervised and unsupervised parser.

For the unsupervised parser, the major issue is that the parses are generally of lower quality. As an illustration, Figures 3.2 and 3.3 represent unsupervised and supervised parses of the same sentence (common edges are indicated in bold). The supervised parse looks relatively reasonable, while the unsupervised parse makes some odd choices (*mr.* being a dependent of the root, the verb having no special status, etc), and contains many adjacent dependencies. Figures 3.4 and 3.5 represent another pair of parse trees, this one with a few longer dependencies than the previous pair.

We take two approaches to finding higher quality individual dependencies for use as features: one is the calculation of fine-grained accuracy rates for dependencies, and the other is Information Gain (IG). In the case of fine-grained accuracy rates, we are assessing which dependencies are likely to be reliable, by comparison with a gold standard,¹⁰ and in the case of IG, we identify which dependencies are particularly strongly associated with positive or negative examples.

3.3.3.2 Unsupervised with Reliability-based Selection

To measure unsupervised dependency reliability, raw precision / recall scores would capture some of that notion — e.g. VBD TO, with 6593 gold-standard instances and 1520 correctly identified (recall = 0.231), is almost certainly more reliable than NNP IN with 7031 gold standard instances and only 547 correctly identified (recall = 0.078) — but they would obviously overrepresent low frequency dependency pairs, which are likely to be particularly unreliable. Taking the precision and recall scores as probabilities of correctness, we therefore adopted a prior α to smooth the scores. Given the relevant denominators N_p for precision or N_r for recall for each dependency pair type, our modified scores use $N_p + \alpha$ and $N_r + \alpha$ respectively; this leaves a probability mass $\frac{\alpha}{\alpha + N_p}$ (resp. $\frac{\alpha}{\alpha + N_r}$) for unseen instances of each dependency. By inspection of the actual raw scores on the training set, we chose $\alpha = 20$. We then select features with modified scores above various thresholds.

For lexicalised dependencies, with their much greater data sparsity issues, we based the choice on the corresponding unlexicalised dependency: if the parts of speech of the lexicalised dependencies matched an unlexicalised dependency above a particular threshold, we selected that lexicalised dependency. (So *the man* would be deemed reliable if *DT NN* were deemed reliable.)

¹⁰The Penn Treebank dependency gold standard was derived using http://nlp.cs.lth.se/software/treebank_converter/.

3.3.3.3 Unsupervised with IG Selection

We calculate IG over the training set, using a standard formulation of Yang and Pedersen (1997):

$$\begin{aligned}
 IG(r) = & - \sum_{i=1}^m \Pr(c_i) \log \Pr(c_i) \\
 & + \Pr(r) \sum_{i=1}^m \Pr(c_i|r) \log \Pr(c_i|r) \\
 & + \Pr(\bar{r}) \sum_{i=1}^m \Pr(c_i|\bar{r}) \log \Pr(c_i|\bar{r})
 \end{aligned}$$

with r representing a dependency, c a binary class, and $m = 2$. We then select features with IG scores above various thresholds. The application to unlexicalised features is straightforward: IG is applied to the unlexicalised dependencies, and some subset of these chosen based on the resulting ranking and a particular threshold. For lexicalised dependencies, there are two alternatives. One is to apply IG to the lexicalised dependencies directly (**direct**); the second is the same as the approach for unsupervised dependencies with reliability selection above, where we extract all the lexicalised dependencies that have corresponding selected unlexicalised dependencies (**indirect**).

Of course, both of these refinements of unsupervised parse features require some kind of annotation, and we discuss the implications for our scenario later in Section 3.4; but here we are just interested in the question of the extent to which unsupervised parsers can produce dependencies that are at all useful for realisation ranking.

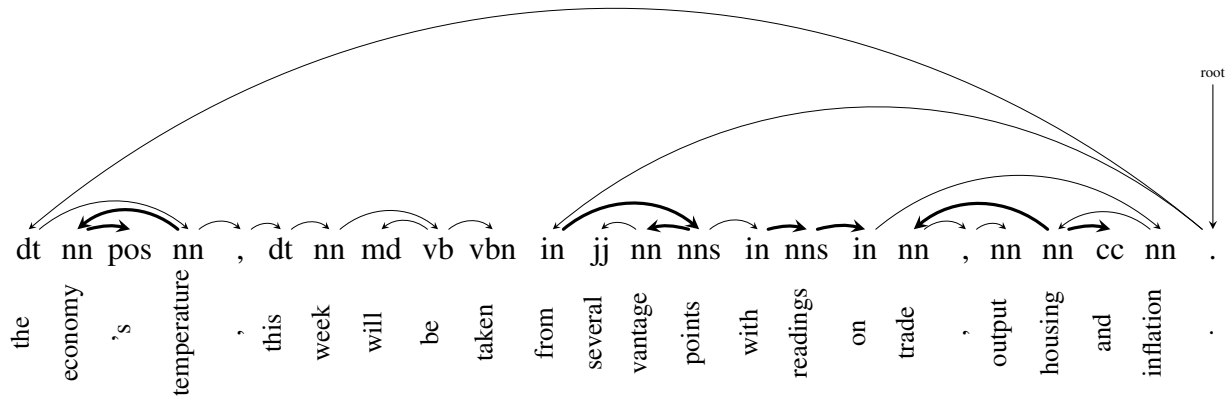


Figure 3.4: Unsupervised dependency tree for another sample sentence. Edges in bold are the dependencies that are identified by both supervised and unsupervised parser.

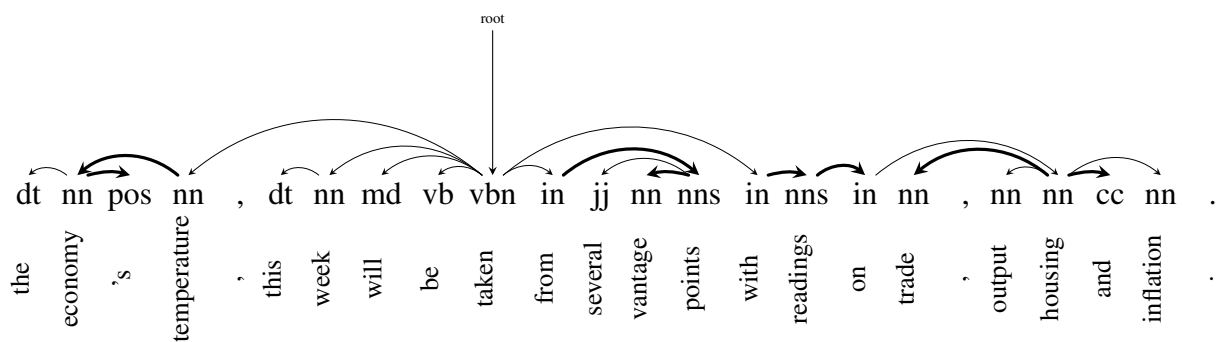


Figure 3.5: Supervised dependency tree for sample sentence of Fig 3.4. Edges in bold are the dependencies that are identified by both supervised and unsupervised parser.

3.4 Results

Base models (supervised) Table 3.3 gives classification accuracy results for the large LMs, and for the supervised parsers, both separately and in combination with the LM, all on the *random* test set (see Section 3.3.3.1). We incorporated LM score as an additional feature to the structural feature set to build *+l-lm4* and *+l-lm2*. The key results here are:

1. Using parse features alone almost always outperforms the LM (except in the case of *dep-unlex-unnamed*), and the combination with the LM always improves over just parse features alone. This is broadly in line with the findings discussed in Section 2.2.1.2 on using structural features from symbolic grammars, and suggests that using external statistical parsers is a valid alternative approach to carrying out realisation ranking.
2. While the *l-lm4* LM forms quite a strong baseline, the *l-lm2* version is somewhat weaker. It does still, however, contribute to an improvement in performance when added to the dependency feature models, of around 2%. This is not unexpected, as it would be contributing information that is complementary to the dependencies, which (in the cases where the dependencies are not adjacent) give longer-distance information. For the same reason, adding this bigram data to the production rule models produces a much smaller improvement.
3. Production rule features are better than dependency features; presumably one reason for this is that there are more production rule features (i.e. the ones consisting of only non-terminal nodes in the tree).
4. The C&J parser’s templates — one particular set of choices for representing compound structural features — outperform just production rules combined with a LM, quite substantially. This also fits with previous work on using compound features.
5. In comparison with the results for the *greatest* test set (Table 3.4), as mentioned above, *random* is always consistently lower, for each comparable cell in the table. This is expected on the assumption that our MED method for choosing positive and negative examples is an accurate reflection of their goodness with respect to the original sentence. There is generally the same pattern for subsequent results as well, so henceforth we only report *random*, as the more conservative of the two measures and as a more realistic scenario (comparing a reference sentence against some arbitrary one, not one that we know is the ‘worst’ of a set of candidates).

model	acc.%	+l-lm4	+l-lm2
l-lm4	68.36	-	-
l-lm2	64.16	-	-
C&J	90.50	-	-
prod-rule-lex	79.48	80.76	80.33
prod-rule-unlex	71.49	72.24	71.64
dep-lex-named	72.45	76.18	74.68
dep-lex-unnamed	71.51	75.54	74.72
dep-unlex-named	69.88	72.92	71.55
dep-unlex-unnamed	63.41	66.75	65.04

Table 3.3: Classification scores for supervised parse features and large LM on *random*: accuracy on parse features; parse features plus large l-lm4 LM over these sentences; parse features plus large l-lm2 LM over these sentences

model	acc.%	+l-lm4	+l-lm2
l-lm4	71.83	-	-
l-lm2	65.19	-	-
C&J	91.61	-	-
prod-rule-lex	84.25	84.45	84.19
prod-rule-unlex	75.81	76.56	76.01
dep-lex-named	76.78	81.11	78.58
dep-lex-unnamed	75.75	80.42	77.98
dep-unlex-named	73.25	75.75	74.38
dep-unlex-unnamed	66.80	69.84	66.97

Table 3.4: Classification scores for supervised parse features and large LM on *greatest*: accuracy on parse features; parse features plus large l-lm4 LM over these sentences; parse features plus large l-lm2 LM over these sentences

model	overall acc.%	#sent	acc.%
unsuper-lex	62.67	941	65.73
unsuper-unlex	58.26	791	62.20

Table 3.5: Classification scores for unsupervised parse features and large LM on *random*: accuracy on parse features; number of sentences where feature vectors differ for positive and negative examples; accuracy over effective test set.

model	#sent	l-lm4	+l-lm4	l-lm2	+l-lm2
unsuper-lex	941	75.29	70.03	69.65	68.97
unsuper-unlex	791	75.53	67.95	71.49	67.95

Table 3.6: Classification scores for unsupervised parse features and large LM on *random*: number of sentences where feature vectors differ for positive and negative examples; accuracy of language models (individually, or combined with unsupervised model) over effective test set.

model	#sent	s-lm4	+s-lm4	s-lm2	+s-lm2
unsuper-lex	941	67.80	68.76	67.48	68.76
unsuper-unlex	791	67.95	64.48	67.48	63.84

Table 3.7: Classification scores for unsupervised parse features and small LM on *random*: number of sentences where feature vectors differ for positive and negative examples; accuracy of language models (individually, or combined with unsupervised model) over effective test set.

Base models (unsupervised) Table 3.5 gives results when unsupervised parse structures are combined with a large LM, see Section 3.3.3.1. Not surprisingly, these results are quite a lot worse than for the supervised: a drop of around 13% for lexicalised dependencies and 8% for unlexicalised. These are also lower than the LMs in Table 3.3. For a more detailed look, we calculated the classification accuracy over those sentences where the feature representation differed (which we refer to as the *effective test set*) and hence the model makes a genuine prediction,¹¹ to see whether a back-off model would be appropriate: if the accuracy for the unsupervised models is higher than for the LM over the effective test set, the decision for that subset could be based on the unsupervised model decision, with a back-off to the LM. Since the effect test set differs from one model to another, numbers can’t be directly comparable. Table 3.5 includes the effective test set, and the unsupervised classification accuracy scores over that set, for each model.

The LMs may also differ over the effective test sets: we present the classification accuracies for these in Table 3.6 (for the large LMs) and Table 3.7 (for the small LMs). The tables include both the accuracy of the LMs alone (e.g. the column *l-lm4*), as well as the accuracy of the LM in combination with the unsupervised model (e.g. *+l-lm4*). It is apparent that the effective test sets for the unsupervised models are also in fact easier for the large LMs: their scores on these subsets are all higher than the overall LM accuracies. They are also higher than the unsupervised models, and the combinations are lower than for the LMs alone. The story is different for the small LMs: while the LMs are higher than the unsupervised models, the combinations are better than both, by 3% in the lexicalised case. The

¹¹Recall that in cases where the model cannot make a prediction, it chooses at random.

conclusion here would be that if a very large LM is available, raw unsupervised parse features would not be helpful; but if only a small LM is available, they would still contribute.

Unsupervised with reliability-based selection Table 3.8 presents the results for our recall-based reliability measure (presented in section 3.3.3.2) along with the effective test set sizes for four thresholds across dependencies with a positive reliability score.¹² These are uniformly poor, and in fact marginally worse than the raw unsupervised features in Table 3.5, so we do not present combinations with the LMs. We looked at the 10 highest- and 10 lowest-ranked features under this measure (Table 3.9), along with their raw counts in the training corpus. The highest-ranked ones were believable as reliable instances of dependencies: infinitival *to* in VB TO seems likely to be often correct, as does existential *there* in the first three cases. However, it is quite possible that many of the reliable ones such as PRP VBZ are actually poor at distinguishing between positive and negative examples by virtue of their frequency — they may occur equally often with both, in the same way that words like *the* are useless in general text classification.

Another possibility is that our reliability metric is not capturing the right phenomenon: that it is flagging as errors systematic intentional choices that the unsupervised parser is making, just because they happen to disagree with the systematic choices of the gold standard. For example, in Figures 3.4 and 3.5, the unsupervised parse contains the dependency MD VB, choosing serial attachment of auxiliaries and modals to the main verb, while the supervised parser contains MD VBN, choosing attachment of all to the main verb instead. Here the unsupervised parser does not necessarily seem incorrect. Indeed, the very smallness of the proportion of correct instances in the lowest-ranking dependency pairs in Table 3.9 suggests that these are not just random (and almost always incorrect) choices by the unsupervised parser; four of them in fact appear to relate to the sort of verb attachment choices mentioned. The issue of the difficulty of comparing dependency treebanks in general, perhaps because of fairly arbitrary decisions about headedness, is raised in Zeman et al. (2012); this would appear to be relevant to the task here too.

¹²This consequently does not include dependencies with recall-based reliability zero. We do not present the precision-based ones here.

model	cut-off%	acc.%	#sent
unsuper-lex	100	65.19	915
unsuper-lex	75	64.33	897
unsuper-lex	50	64.37	856
unsuper-lex	25	64.55	708
unsuper-unlex	100	61.44	721
unsuper-unlex	75	62.39	686
unsuper-unlex	50	62.00	629
unsuper-unlex	25	62.44	442

Table 3.8: Classification scores for unsupervised parse features selected by reliability on random: threshold cut-off accuracy on parse features; number of sentences where feature vectors differ for positive and negative examples (effective test set)

Highest			Lowest		
feature	correct	# in gold	feature	correct	# in gold
EX VBZ	72	179	VBZ VBP	1	201
EX VBP	45	110	VBP VBP	1	218
EX VBD	27	72	VBD NNP	1	292
DT VBZ	108	364	VBP VBD	1	335
\$ TO	151	541	VBZ VBD	2	696
PRP VBZ	509	1792	IN CD	2	860
RP VBN	60	222	NN \$	1	481
PRP VBD	600	2423	CD RB	1	485
VB TO	1520	6593	CD TO	1	487
NNS JJ	28	103	IN MD	1	694

Table 3.9: Highest- (left) and lowest- (right) ranking dependency features based on reliability measure. For each, columns represent the dependency; the number of times it was correct in an unsupervised parse; and the number of times it occurred in the gold standard.

model	cut-off%	acc.%	#sent	l-lm4
unsuper-lex (direct)	100	71.97	710	80.28
unsuper-lex (direct)	75	82.17	300	81.17
unsuper-lex (direct)	50	83.26	242	80.58
unsuper-lex (direct)	25	90.00	185	82.70
unsuper-lex (indirect)	100	74.57	920	78.80
unsuper-lex (indirect)	75	82.56	894	78.91
unsuper-lex (indirect)	50	85.19	849	79.62
unsuper-lex (indirect)	25	89.53	717	79.50
unsuper-unlex	100	60.84	738	79.20
unsuper-unlex	75	60.59	708	79.45
unsuper-unlex	50	61.08	657	80.29
unsuper-unlex	25	62.06	506	80.90

Table 3.10: Classification scores for unsupervised parse features selected by IG: threshold cut-off; accuracy on parse features; number of sentences where feature vectors differ for positive and negative examples (effective test set); large LM score over these sentences

rank	features				
1–5	TO VB	VBN VB	VB VBN	VB TO	VBG VB
6–10	VB VBG	NNS TO	IN RBR	VBN TO	DT RP

Table 3.11: Highest ranking features based on IG.

Unsupervised with IG selection Table 3.10 shows the results for selecting the top $k\%$ unsupervised parse features ranked by positive IG scores (described in Section 3.3.3.3).¹³ We see a dramatic improvement in the lexicalised case (*direct*) over the raw features (Table 3.5). The effective test set also falls a lot more for the *direct* lexicalised case; this is not surprising, as the top (say) 25% lexicalised features on the training set are much less likely to occur in the test set than the top 25% unlexicalised features. On the other hand, the *indirect* lexicalised case — where lexicalised features are instantiated on the basis of IG-ranked *unlexicalised dependencies* — have a sentence coverage that is much greater, as would be expected, but perhaps surprisingly an accuracy that is comparable to the *direct* method, and in fact even higher for three of the four thresholds presented in Table 3.10. For the 25% threshold on the *indirect* method, the model performs around 10% higher on around 60% of the total test set, making a back-off model a very suitable option.

Table 3.11 presents the top 10 unlexicalised features. It is not immediately apparent why there is a preponderance of paired verbs, such as VB VBG or VB TO (with the infinitival *to* probably indicating another verb following). Perhaps verb sequences indicate poor sentences, although this would require further inspection of the data.

Illustration of Generated Content To see how unsupervised parses, though bad, might contribute useful structure, we present an example where the unsupervised parse model predicted the better candidate, while the other models did not. The original sentence is given in (3.1), the preferred candidate in (3.2), and the dispreferred candidate in (3.3). The preferred candidate is basically the same as the original, slightly odd punctuation notwithstanding, while the dispreferred candidate has obvious problems in terms of ordering.

- (3.1) For example, their selling caused trading halts to be declared in USAir Group, which closed down 3 7/8 to 41 1/2, Delta Air Lines, which fell 7 3/4 to 69 1/4, and Philips Industries, which sank 3 to 21 1/2.
- (3.2) For example their selling, caused trading halts to be declared in USAir Group which closed, down 3 7/8 to 41 1/2 Delta Air Lines which fell 7 3/4

¹³That is, those cases with an IG of zero — i.e. entirely non-distinguishing between positive and negative examples — are excluded, which is why the 100% scores are higher than for the raw features of Table 3.5.

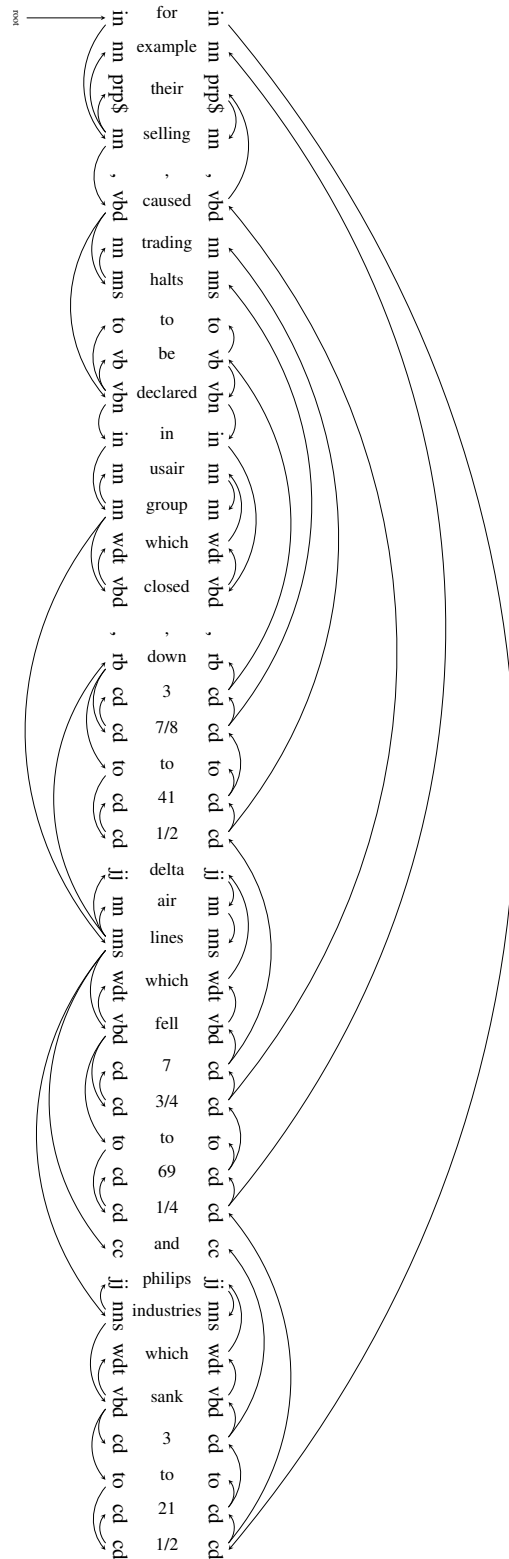


Figure 3.6: Supervised dependencies (lower arcs) vs unsupervised dependencies (upper arcs) for sentence 3.2

to 69 1/4 and Philips Industries which sank 3 to 21 1/2.

- (3.3) For example to be declared in USAir Group, their selling, caused trading halts which closed, down 3 7/8 to 41 1/2 Delta Air Lines which fell 7 3/4 to 69 1/4 and Philips Industries which sank 3 to 21 1/2.

Figure 3.6 shows the unsupervised (upper) and supervised (lower) parses for the preferred candidate (3.2). For the most part, the supervised parser makes sensible linguistic choices: it identifies as the head of e.g. *Delta Air Lines which fell 7 3/4 to 69 1/4* the final noun of the named entity, which in turn is a dependent of the head of the previous parallel clause; the unsupervised parser, by contrast, has as the ultimate head the final numeral, with odd choices in the intermediate edges as well such as *which* being the head of the named entity in each case. However, the supervised parser makes a fundamental error in grouping two chunks of numerical quantities (i.e. *down 3 7/8 to 41 1/2* and *fell 7 3/4 to 69 1/4*) being associated with the same named entity. The unsupervised parse does not do this: while it makes odd choices, they are consistent odd choices. This consistency of choices may in fact not be accidental, and is further reason to reconsider our approach to assessing the reliability of unsupervised parse edges.

Implications If the reliability-based selection for unsupervised parse features had proved effective, it would have been necessary to find some mechanism to approximate a check against a gold standard, such as constructing a committee of unsupervised parsers and using only those dependencies that received a sufficient number of votes. However, this is not warranted by the results.

For the IG-based selection, what is necessary is a binary annotation of sentence pairs as preferred or dispreferred. This is much less intensive than treebank annotation, and so a reasonable alternative to constructing supervised parsers. As noted, a large LM outperforms raw unsupervised features; but even constructing large corpora (semi-)automatically for many of the world's languages, as per Scannell (2007), is challenging, and the IG-selected features in any case do better than the large LM alone and better still in conjunction with it.

3.5 Summary

For symbolic grammars that are small and/or in development, there may well not be many resources to draw on for building good realisation ranking models for natural language generation. In this chapter we have examined unsupervised parsers as a possible source of structural features for such models, notwithstanding their generally much poorer quality of parses than supervised parsers.

We found that they can indeed be useful. In the general case, unsupervised parse features contribute in the case of smaller language models, of the sort that might be available for many less resourced languages. They also contribute very strongly over reasonable sized subsets of the test set once useful features have been identified by Information Gain: there are improvements of up to 10% in classification accuracy over 60% of the test set, making a model that uses unsupervised features and then backs off to a language model an attractive option. This would be feasible in scenarios where it is possible to annotate pairs of sentences as preferred and dispreferred.

Choosing features by a reliability-based measure did not prove useful. However, this may be related to systematic choices made by the unsupervised parser that were different from the gold standard's choices, rather than bad parsing; an option for aligning systematic choices is the HamleDT approach and software of Zeman et al. (2012). In addition to the unsupervised parser that is used in this chapter, Naseem (2014) introduces two other linguistically motivated models that seem to be effectively performing for the languages which only a light level of supervised training can be achieved. For computationally resource-poor languages, transfer dependency parsers can be regarded as promising alternative to the unsupervised parser and has gained increasing attention in the field. To recap we briefly describe one of the prominent works in that area. McDonald et al. (2011) showed that training parsers on delexicalised data can be transferred between languages and produce higher accuracies compared to existing unsupervised parsers. Their finding allows minority languages who lack parser and treebanks to have access to remarkable parsing technology. With their direct transfer, as the name implies, they simply mapped PoS tags from the language that parser was originally trained on to the target language. However, their projected transfer parser requires a large parallel corpus to perform parsing, starting from the direct transfer algorithm and gradually learn to parse lexicalised sentence in the target language. The authors also showed that using multiple languages as the source of learning would improve the overall quality of the parse trees for the target language. One consideration about their proposed approach, as raised by Täckström et al. (2013) was that it worked well for similar languages. To address this shortcoming, Täckström et al. (2013) proposed selective parameter sharing based on typological and language-family features.

Another option for improving performance is the use of compound features, as is used in much of the work discussed in Section 3.2. The results in this chapter for the supervised parsers showed that the model with compound (or 'higher order') features dramatically outperformed the ones with simple features; it could be promising to extend these to the dependency models, as for example in their use in parse reranking, such as by Hall (2007) and Wang and Zong (2011). In addition to evaluating the approach by measures other than binary classification accuracy (e.g. BLEU or the ranking score of Cahill et al. (2007b)), and examining other potential comparators (e.g. the treebank-trained generator of Belz (2005)), the next major

step would be applying it to one of these smaller languages that has motivated the work in this chapter.

Now that we have established that structural features from outside a symbolic grammar — specifically, from both supervised and unsupervised statistical parsers — can be useful in realisation ranking, the next step is to see how they perform in combination with features derived from the symbolic grammar: are they complementary?

Chapter 4

Combinations with Internal Structures for Reranking

4.1 Introduction

In this chapter, we detail the hybrid reranking models that use system internal representations alone or in conjunction with statistical parser features. The objective of this part of research is to study the contribution of features extracted from existing internal structures to reranking quality and whether combining these features with features from external statistical parsers promotes reranking quality.

To recap from Section 2.2.1.2, system internal representations are intermediate structures produced by language processing systems in the text analysis/generation process. Since the Xerox Linguistics Environment (XLE) is the text generation engine in this research, Lexical Functional Grammar (LFG) primary structures—constituent structure (c-structure) and functional structure (f-structure)—are the intermediate structures.

In this chapter, we combine the work of Cahill et al. (2007b) that motivates using system-internal representations for reranking, with the approaches that use external resources to extract representations similar to the ones derived from statistical parsers and proved to be successful in Chapter 3. We investigate our approach over two grammars that are in two different stages of development. The first grammar is mature and almost comparable to the one used in Cahill et al. (2007b). The results over this grammar can confirm the positive contribution of internal representations in reranking as previously concluded by Cahill et al. (2007b) over German. The second grammar is a basic grammar that only specifies the right branching factor. Building models using these grammars allows us to study the effect of grammar maturity/size in reranking and compare the contribution of features from statistical parsers when combined with features from either of these grammars. To be more

precise, we would like to see if the external parsers capture additional information to the manual grammar, i.e., to what extent features from a statistical parser can enhance reranking when combined with features from the manual grammar. If the answer to this question is positive, we would further investigate the effect of grammar size on the reranking quality. In Section 4.2, we introduce the proposed models for reranking using features from the internal representation and their combination with statistical parser features. Finally the experiment configurations and experimental results are discussed and compared with Cahill et al. (2007b)’s result in Section 4.3. We also discuss whether the conclusion that Cahill et al. (2007a) made for German can be applied to English.

4.2 Features from Intermediate Structures

Reranking features are the distinguishing elements between various models we use for reranking. We are using mostly the same features as Cahill et al. (2007b): when we use only internal structures, we are broadly replicating their work. The aim is to compare this with the hybrid models that include features derived from external structures. These features are extracted from the c-/f-structure of the alternatives regenerated from a given gold sentence using XLE. We introduce these reranking in the next two subsections followed by an overview of how each model is set-up.

4.2.1 C-structure Features

In Chapter 3, the best reranking result was achieved by applying features from the C&J parser. The high accuracy is likely to be somehow related to the usage of the various templates that are used in the parser’s reranking process. Each template captures some bit of information and combining them together will result in a model that has a better representation of correct word order, given the internal structure. In this part of the research, we extend our models from only production rules to hybrid models. Inspired by templates deployed in previous works by Charniak and Johnson (2005), Cahill et al. (2007b), we introduced the following four c-structure features as part of the proposed hybrid reranker:

PARENT & CHILD (CS-PC) counts the occurrences of parent-child instance.

This feature proved to be helpful in realisation reranking as discussed in the previous chapter. PAREN-P is an example of parent-child relationship that is highlighted in Figure 4.1(a).

GRANDPARENT, PARENT & CHILD (CS-GPC) counts the occurrences of three nodes descending from one another directly, e.g., *NP_{adj}-NP_{zero}-PAREN-P* triple in Figure 4.1(b) is an instance of CS-GPC.

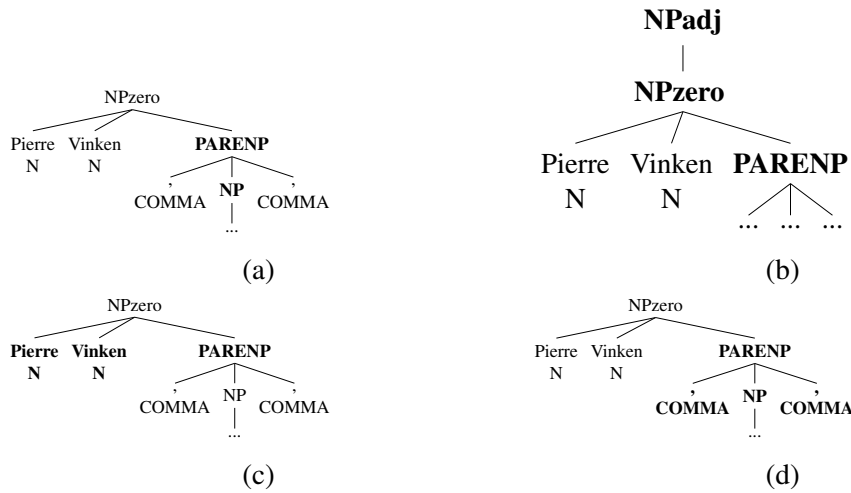


Figure 4.1: Examples of the realisation reranking c-structure features; subtrees are extracted from the c-structure presented in Figure 2.7: (a) CS-PC, (b) CS-GPC, (c) CS-SB, and (d) CS-NC.

SIBLINGS (CS-SB) counts the occurrences of a set of nodes that all descend from a single node. There is no limit on the number of the children. *Pierre-Vinken-PAREN* represents such feature in Figure 4.1(c).

PARENT & CHILDREN (CS-NC) counts the number of CS-SB that descend from a certain parent node. *NPzero-Pierre-Vinken-PAREN* and *PAREN-PAREN-NP-*, would be the CS-NC features in the subtree in Figure 4.1(d).

The last three tree features are comparable to the **Rule** schema from the **C&J** reranking parser. These four features are designed so that they can encode information provided by c-structure in various levels: CS-PC captures single-level vertical structure of the tree. CS-GPC extends CS-PC and looks into vertical slices of depth of two. CS-SB captures horizontal slices of the structure at all levels of the analysis, where each slice is defined as a set of nodes that descend from one parent. This feature is contributing to the ordering of nodes: words or nodes' labels, depending on the depth of the slice, at a phrase level. CS-NC in a sense combines CS-GPC with CS-SB as it captures the horizontal and vertical slices at the same time. This combination of terminals and non-terminals allows the model to learn about the structure and word order at the same time.

4.2.2 F-structure Features

There is a structural difference between the research carried out by Cahill et al. (2007b) and this research: ranking vs reranking. The former ranks the alternative

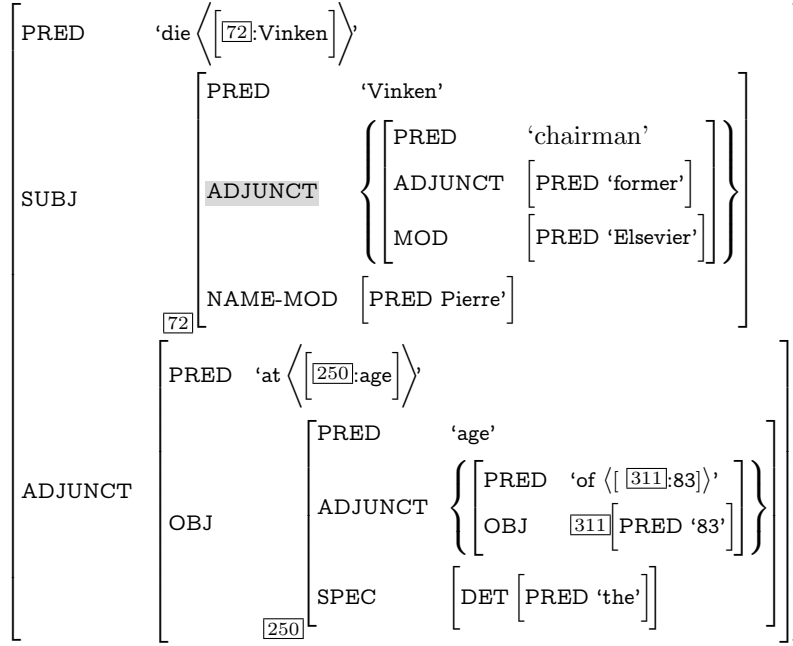


Figure 4.2: FS-ATTRS

strings just by regenerating them from the unique f-structure they all share. However, in our work we assume that the reranker may be an independent module (so as to incorporate external features). Therefore, it analyses the candidates from scratch, i.e., parsing them to get the c-structure and f-structure. Reparsing reduces the likelihood of having a unique f-structure for all candidates regenerated from one string. Consequently, f-structure becomes a potential feature source. In this section, we list the f-structure features that we use for reranking. These features were previously used for parse ranking but not for realisation ranking by Cahill et al. (2007b).

ATTRIBUTES (FS-ATTRS) is a feature template that counts the number of occurrences of a given attribute within an f-structure. ADJUNCT is an example of such an attribute in Figure 4.2 with 4 occurrences, so the feature value pair for ADJUNCT looks like $\text{FS-ATTRS-ADJUNCT } 4$. FS-ATTRS can take multiple f-structure attributes as a parameter, rather than just one. Similar to Cahill et al. (2007b)’s work we only consider one attribute at a time. There is no obvious process a priori for how attributes should be grouped in order to form an informative set of parameters for FS-ATTRS.

ATTRIBUTE-VALUE PAIRS (FS-ATTR-VAL) takes two parameters: an f-structure attribute and a potential value for this attribute. This template can be regarded as an extended version of CS-PC, since it captures more than terminal production rules, i.e. the ones that do not exist in c-structure. For

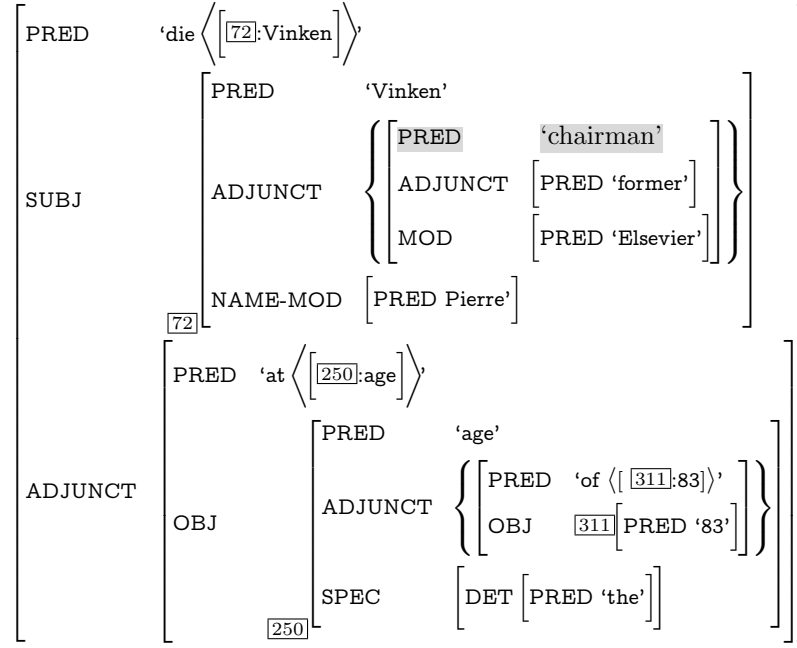


Figure 4.3: FS-ATTR-VAL

instance, `FS-ATTR-VAL-TENSE-past` counts how often the atomic attribute `TENSE` has the value `past`. Figure 4.3 highlights the bits that form the `FS-ATTR-VAL-PRED-chairman` feature.

ADJACENT ATTRIBUTES (FS-ADJ-ATTR) is another template extracted from an f-structure. It is the f-structural equivalent of non-terminal production `CS-PC`. It takes two f-structure attributes as parameters, and counts how often the first (necessarily complex) attribute directly embeds the second attribute (which can be either complex or atomic). In Figure 4.4, `FS-ADJ-ATTR-ADJUNCT-PRED` is a feature that occurred 4 times.

SUB-ATTRIBUTES (FS-SUB-ATTR) feature template is a generalised/non-local variant of `FS-ADJ-ATTR`. Similar to `FS-ADJ-ATTR` it accepts two f-structure attributes as parameter but the first attribute can indirectly embed the second attribute. As highlighted in Figure 4.5 `FS-SUB-ATTR-OBJ-DET` is an instance of this template.

AUNT ATTRIBUTES (FS-AUNT-ATTR) feature template represent the relationship between three attributes where the first two attributes are descending from one attribute (siblings) and the third attribute is the direct descendant of either of the first two attributes. In other words, this template extends `FS-ADJ-ATTR` by including the sister of the parent attribute. `FS-AUNT-ATTR-SUBJ-ADJUNCT-OBJ` is an instance of this template highlighted in Figure 4.6

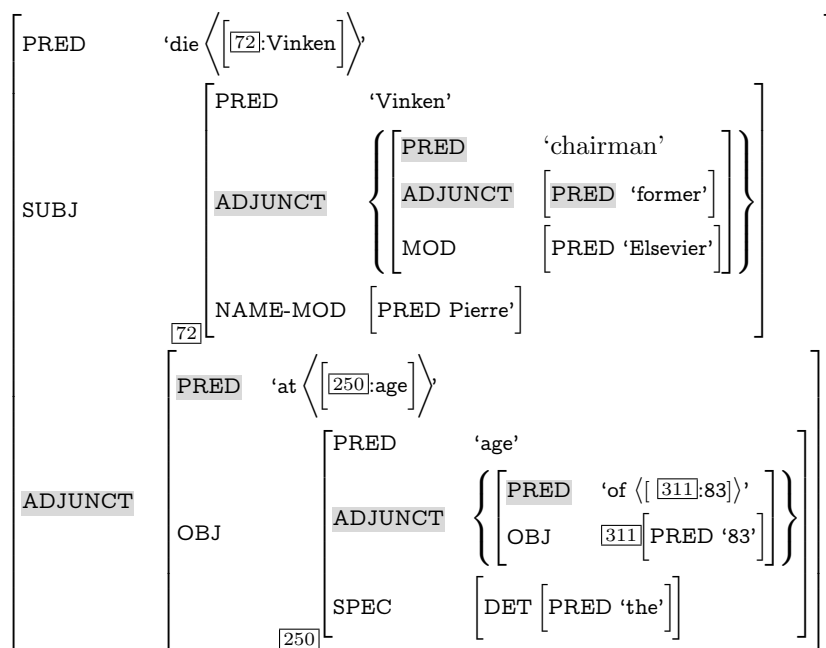


Figure 4.4: FS-ADJ-ATTR

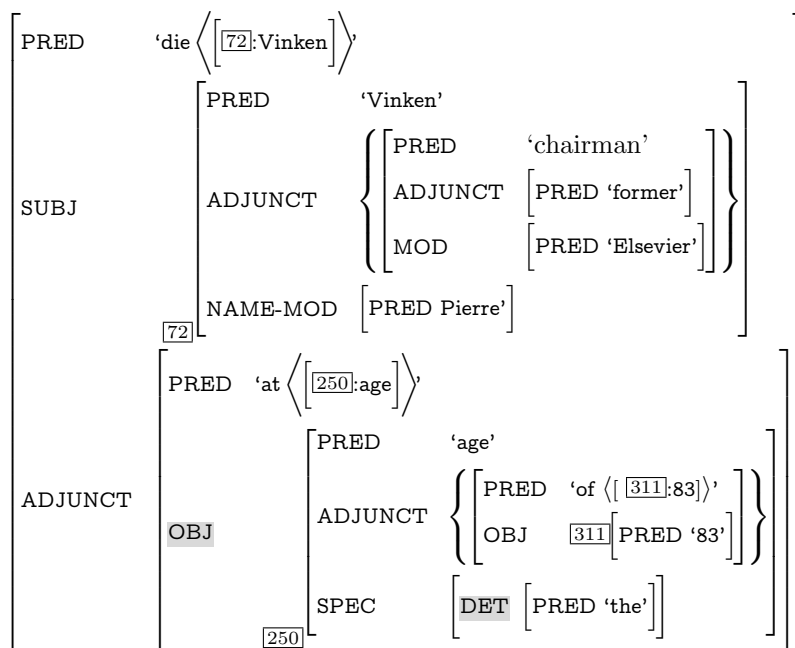


Figure 4.5: FS-SUB-ATTR

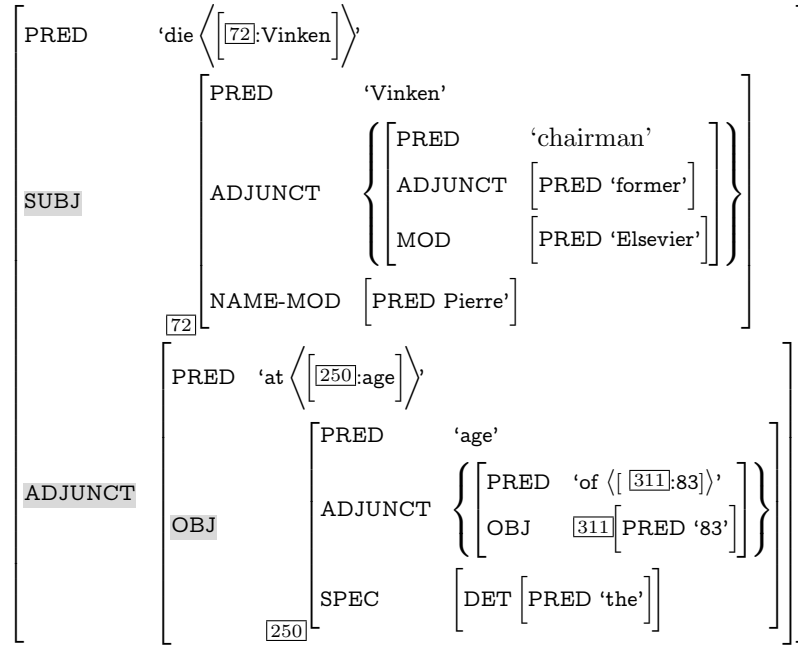


Figure 4.6: FS-AUNT-ATTR

LEXICAL SUB-CATEGORIES (FS-LEX-SUBCAT) is a feature template that counts how often a given verb has one of args sets as arguments. FS-LEX-SUBCAT for *die* is `PASSIVE -`, `STMT-TYPE declarative`, `VTTYPE main`, `LAYOUT-TYPE unspec`. Since this feature is expected to be identical for alternative sentences, its helpfulness will be discussed later in Section 4.3 .

VERB ARGUMENTS (FS-VERB-ARGS) is another feature template that specifically looks at verbs. This feature captures the verb arguments. As can be seen in Figure 4.8, `FS-VERB-ARGS-die-SUBJ-ADJUNCT` is the only instance of this template since the example contains only one verb. This feature is not expected to discriminate between candidates since the candidates' f-structure generally do not vary in this respect.

Table 4.1 summarises the features described in this sections.

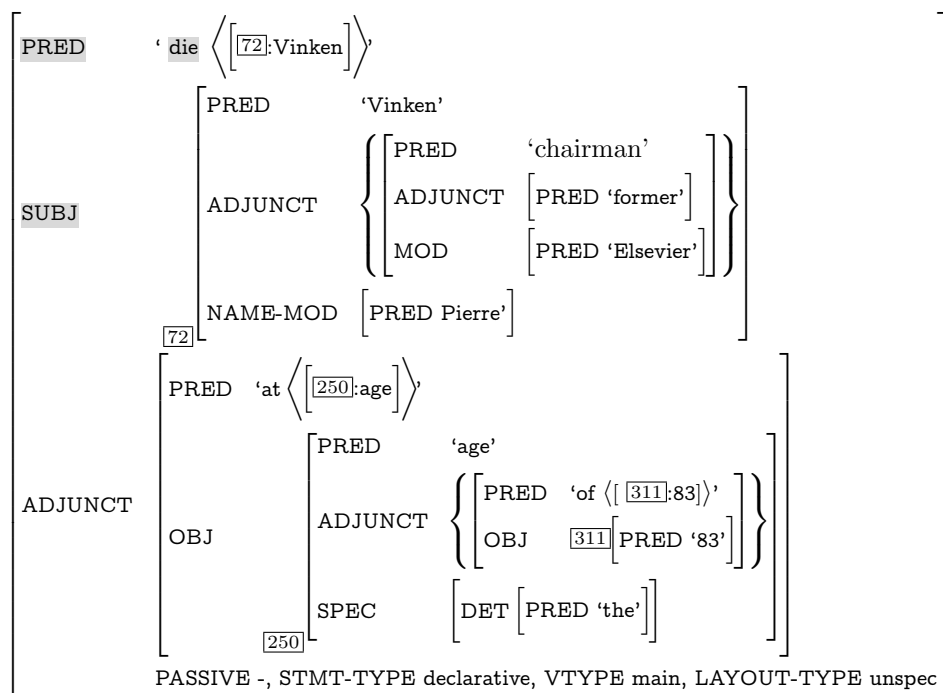


Figure 4.7: FS-LEX-SUBCAT

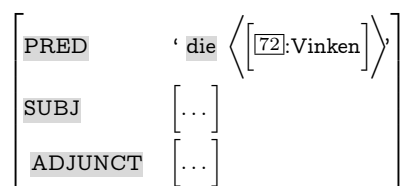


Figure 4.8: FS-VERB-ARGS

Table 4.1: Reranking templates extracted from f-structure features

Name of feature template and parameters	Explanation
c-structure Features	
CS-PC <cat1> <cat2>	counts number of times category <cat1> produces category <cat2> in the given c-structure
CS-GPC <cat1> <cat2> <cat3>	counts number of times category <cat1> produces category <cat2> and category <cat2> produces category <cat3>
CS-SB <cat1> <cat2>...<catn>	counts the number of <cat1> node has nodes <cat2>-<catn> as its sibling
CS-NC <cat1> <cat2>...<catn>	counts number of the times where <cat1> node directly dominates <cat2>-<catn> nodes
F-structure Features	
FS-ATTRS	counts number of occurrences of attribute(s) <attrs> in the f-structure
FS-ATTR-VAL <attr> <val>	counts number of times f-structure attribute <attr> has value <val>
FS-ADJ-ATTR <attr1> <attr2>	counts the number of times feature <attr2> is immediately embedded in feature <attr1>
FS-SUB-ATTR <attr1> <attr2>	counts the number of times feature <attr2> is embedded somewhere in <attr1>
FS-AUNT-ATTR <attr1> <attr2> <attr3>	counts the number of times feature <attr1> and <attr2> are siblings and <attr3> is the direct descendant of either.
FS-LEX-SUBCAT	counts how often a given verb pred has one of args sets as arguments; args are features/values that describe the verb, i.e., Passive feature can be either + or -.
FS-VERB-ARGS	captures and counts verb along with its arguments, i.e SUBJ-ADJUNCT
Language Model Features	
NGRAM SCORE %X	3-gram language model score assigned to the generated sentence

4.2.3 Other Considerations

4.2.3.1 Grammar Size

Since one of the key aspects of this research is finding resources for reranking automatically generated texts in minority languages, we extracted the above mentioned features using two different grammars:

ParGram English Grammar (Large Grammar) which is a very large grammar and available upon request from the ParGram project. This grammar has a very good coverage. All the examples of the c-/f-structures discussed so far have been produced by the Large Grammar.

Starter English Grammar (Small Grammar) that comes with XLE¹. This very basic grammar is mainly used as a template to extend the grammar for other languages.

In contrast to the richly-annotated structures produced by the large grammar, the small grammar produces structures that are generic with almost no sentence-specific analysis. It maps all the sentences to a right branching tree so that all the leaf nodes in c-structure are labelled as TOKEN (versus word specific POS tag in large grammar) and all the internal nodes are labelled as FRAGMENTS (versus grammatical functions such as NPs). The same right-branching approach applies to f-structure. Attributes are either First or REST. First are the attributes that enclose a token (leaf node) and REST attributes comprises a pair of FIRST and REST attributes. This grammar is simple to the point that it does not face much real ambiguity when it comes to regeneration. We note then that results across small and large grammars thus cannot be directly compared: the alternative sentences generated by each are radically different, and for the large grammar, much richer. The small grammar is just to give an idea about how combination of internal and external features might differ across grammar size. If the proposed approach works, the next step would be to build grammars of different sizes by incorporating various subsets of the large grammar English lexicon into the Starter Grammar.

Figures 4.9 and 4.10 depict c-/f-structure for sentence 2.1 produced by the small grammar. Comparing these two structures with their equivalents produced by the large grammar (Figures 2.7 and 2.8) confirm that the small grammar produces much simpler representations than the ParGram English grammar.

¹ or can be downloaded <http://www2.parc.com/isl/groups/nlitt/xle/doc/PargramStarterGrammar/>

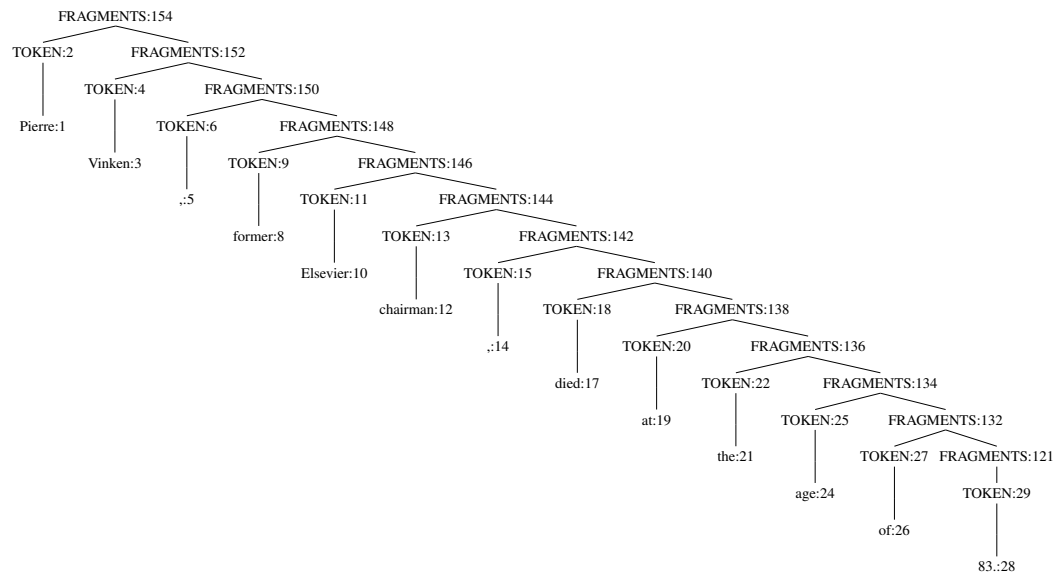


Figure 4.9: c-structure for sentence (2.1) using ParGram Starter Grammar

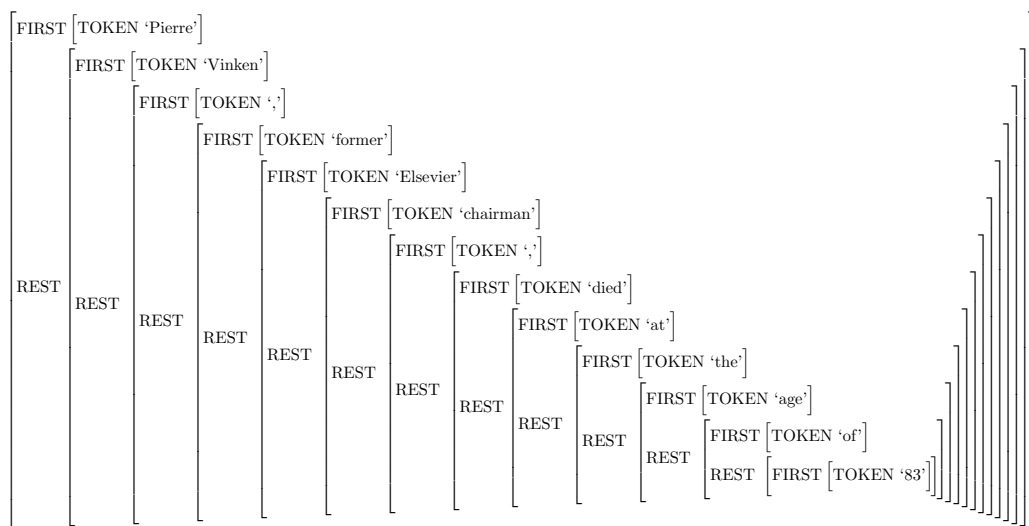


Figure 4.10: f-structure for sentence (2.1) using ParGram Starter Grammar

Model	C-structure features				
	cs-PC	cs-GPC	cs-SB	cs-NC	lx
C1	+	-	-	-	-
C2	-	+	-	-	-
C3	-	-	+	-	-
C4	-	-	-	+	-
C5	+	+	+	+	-
C6	+	-	-	-	+
C7	-	+	-	-	+
C8	-	-	+	-	+
C9	-	-	-	+	+
C10	+	+	+	+	+

Table 4.2: List of model names; + and - represent the presence or the absence of the given feature.

Model	F-structure Features						
	FS-ATTRS	FS-ATTR-VAL	FS-LEX-SUBCAT	FS-ADJ-ATTR	FS-SUB-ATTR	FS-VERB-ARGS	FS-AUNT-ATTR
F1	+	-	-	-	-	-	-
F2	-	+	-	-	-	-	-
F3	-	-	+	-	-	-	-
F4	-	-	-	+	-	-	-
F5	-	-	-	-	+	-	-
F6	-	-	-	-	-	+	-
F7	-	-	-	-	-	-	+
F8	+	+	+	+	+	+	+

Table 4.3: List of model names; + and - represent the presence or the absence of the given feature.

4.2.3.2 Incorporation of Features from Statistical Parsers

In Chapter 3 we looked at a range of statistical parsers and identified their level of contribution to reranking problem. By introducing hybrid reranking models which incorporate various features from statistical parsers with c-/f-structure features and comparing their accuracy with their base models, we aim to answer the following question:

How distinct is the information captured by a statistical parser compared to human authored grammars? Consequently, can they be usefully combined?

4.2.4 Models

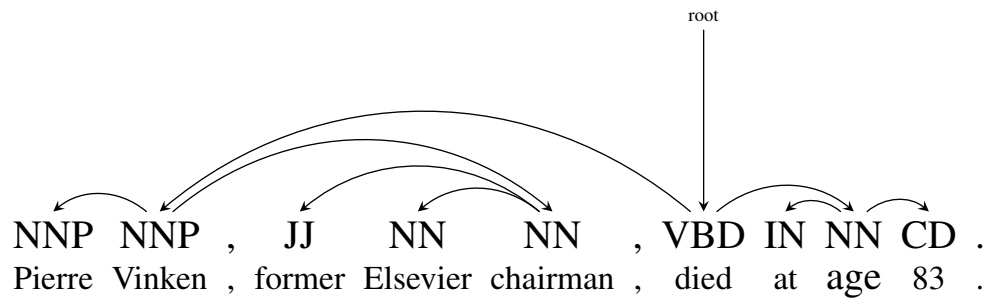
Base and hybrid models with internal structures The ten models listed in Table 4.2 study the contribution of each set of c-structure feature template(s) to the quality of reranking. Similarly, a list of models built upon f-structure templates are introduced in Table 4.3. We named all the models following the same naming convention that reflects which internal structure is present in the model; models that contain letter F deploy f-structure features and models with C deploy c-structure features. These model names will be used later in Section 4.3 for discussion of the experiment results.

Later on, the best performing models regarding their feature category (c-/f-structure) will be picked. The L and S in the model names reflect the features are extracted from the large grammar and small grammar, respectively.

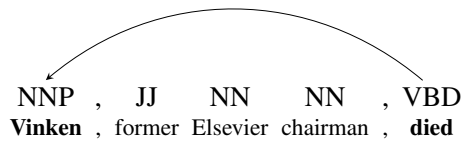
Dependency features for hybrid models To answer the question of whether any reranking-helpful information can be folded in from other resources, we choose four feature template extractable from dependency trees; that's what unsupervised parsers produce, and we can also derive this structure from supervised parsers (see Section 3.3.2). These templates are introduced as a pair or triple of nodes where n_i represents the i th element in the pair. With respect to the model lexicalisation choice, lexicalised or unlexicalised, the elements are either words or Part of Speech (PoS) tags. Figure 4.11 depicts the dependency tree and the four feature templates for sentence (2.1).

1. head-dep (dep): A pair (n_1, n_2) where n_1 is the head of n_2 . In Chapter 3, this feature proved to be helpful in reranking as the sole source of structural information (see Figure 4.11(b)).
2. ancestor-head-dep: A triple (n_1, n_2, n_3) where n_2 is the dependent of n_1 and the head of n_3 . In other words, n_1 is the ancestor of n_3 . In Figure 4.11(c), (died, Vinken, chairman) is an instance of this template.
3. siblings: An n -tuple of words that share the same head word. The pair (former, Elsevier) in Figure 4.11(d) is a valid lexicalised sibling feature.
4. mother-siblings: A head word and all of its dependents without any limitation on the number of the dependents, e.g. the triple (chairman, former, Elsevier) (Figure 4.11(e)) where the word *chairman* is the head word for the other two.

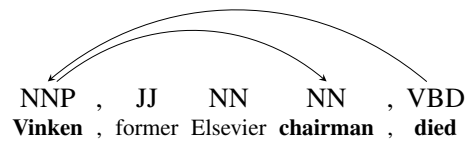
We combine these four dependency features into 7 different models. Considering both lexicalised and unlexicalised versions as we did in Section 3.4, the number of the models is doubled, thus giving 14 in total. These models are listed in Table 4.4. The initial letter, D, in the model name refers to the nature of features used, i.e. dependency features. This naming convention serves as a clue later in identifying feature types present in each hybrid model.



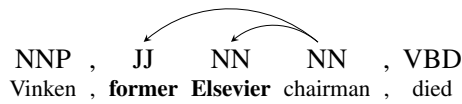
(a)



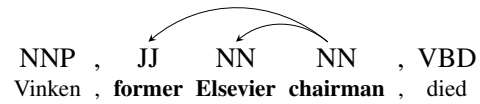
(b)



(c)



(d)



(e)

Figure 4.11: (a) is the dependency tree for sentence (2.1). In (b), (Vinken, died) pair is an example of lexicalised dep feature. The (VBD, NNP, NN) triple in (c) represents an unlexicalised instance of the ancestor template. (d) The lexicalised pair (former, Elsevier) are two siblings that share **chairman** as their mother. Finally, (e) represents the mother-siblings template by the (chairman, former, Elsevier) triple.

Model	dep	ancsstr	sbl	fathrSbl
D1	unlx	-	-	-
D2	-	unlx	-	-
D3	-	-	unlx	-
D4	-	-	-	unlx
D5	unlx	unlx	-	-
D6	unlx	unlx	unlx	-
D7	unlx	unlx	unlx	unlx
D8	lx	-	-	-
D9	-	lx	-	-
D10	-	-	lx	-
D11	-	-	-	lx
D12	lx	lx	-	-
D13	lx	lx	lx	-
D14	lx	lx	lx	lx

Table 4.4: Naming convention for various models using dependency structure

4.3 Experiment Configuration and Experiment Results

In the previous chapter we confirmed the helpfulness of features from statistical parsers in reranking. This chapter’s aim is to answer the following two questions:

RQ4.1 Do features extracted from NLG internal structures contribute to the quality of the reranker? This is essentially the question asked by Cahill et al. (2007b) but with the additional features defined in this chapter for the independent ranking module context.

RQ4.2 Are these features orthogonal to features extracted from statistical parsers? In other words, to what extent would the reranker quality be enhanced if both set of features are deployed.

Prior to answering these questions by analysing the experiment results, we describe the data preparation and experiment setup.

4.3.1 Data Preparation and Experiment Setup

We used the same data and followed the same approach that was described in the previous chapter. To avoid the repetition, we quickly review the configuration from a high level point of view, but a detailed discussion on data and pair selection can be found in Section 3.3.1. We took sections 2-21 of the Penn Treebank for training

and section 23 for testing. Data preparation has been done in the following three steps:

1. Creating alternatives: using XLE to parse the sentences and then using the re-generation module to build the alternatives.
2. Pairing the alternatives and labelling them: The alternative with the highest MED score would be labelled as the positive example. For the negative example, an alternative is randomly selected out of the alternatives with MED score less than the largest MED.
3. Feature extraction: For each model the associated features are extracted using the corresponding grammar and/or parser. For instance, the model which is built upon the internal structures and labelled as small grammar has its features extracted from the small-scale ParGram grammar.

The next step is building a reranking model using MegaM. Here MegaM takes training pairs, in the form of a set of labeled features, as input and returns a model, i.e., a set of weights associated with each feature. Later, this model will be used to assign labels to the test data. Our evaluation metric is the classification accuracy of predicting the correct label for positive and negative example from each pair. We identified the notion of *effective test set* in Section 3.4: this is the set of pairs for which the features differ. In Chapter 3, we concluded that a back-off strategy — where the feature-based reranker is used for the effective test set, and the n-gram language model for the remainder — could improve performance. In this chapter, we thus present the overall accuracy under this back-off strategy. We give the overall accuracy (T) as per definition (4.1) to study each feature set contribution to reranking.

$$T = (T_e * T_{e\text{cov}}\%) + (T'_e * T'_{e\text{cov}}\%) \quad (4.1)$$

where T_e is the effective test set, $T_{e\text{cov}}\%$ is the coverage percentage and $T'_e = 1 - T_e$. As for the back-off strategy, we use our baseline language model (LM), *sl-lm2* that was previously introduced in Section 3.4.

4.3.2 Experiment Results

4.3.2.1 Base Models

Base models are designed to answer RQ4.1 asked on page 91 of this chapter.

C-structure models Table 4.5 presents the coverage, effective test set and overall accuracy for the models that are built just upon c-structure features from both large-scale (Table 4.5 (a)) and small-scale (Table 4.5 (b)) grammar. These models were

Model	T_e		
	cov%	acc.%	acc.%
CL1	85.9	85.37	84.18
CL2	90.97	83.73	82.46
CL3	90.97	79.56	78.67
CL4	90.97	79.56	78.67
CL5	92.86	83.41	82.03
CL6	86.59	86.73	85.77
CL7	91.83	85.52	84.57
CL8	90.54	79.28	78.46
CL9	90.54	79.28	78.46
CL10	93.64	84.97	83.88

(a)

Model	T_e		
	cov.%	acc.%	acc.%
CS1	58.56	87.21	80.88
CS2	58.56	87.06	80.79
CS3	57.7	88.21	81.48
CS4	57.7	88.21	81.48
CS5	58.56	87.06	80.79
CS6	69.99	95.45	89.73
CS7	69.99	95.45	89.73
CS8	57.7	88.21	81.48
CS9	57.7	88.21	81.48
CS10	69.99	95.57	89.81

(b)

Table 4.5: Results of reranking with c-structure features extracted from the large grammar (a), and the small grammar (b).

previously introduced in Table 4.3. As can be seen in that table, C6-C10 models are the lexicalised version of C1-C5; this explains why C_i and $C_i + 5$, where $i \in \{1, 2, 3, 4, 5\}$, are always paired up in the analysis of the results.

The large-scale grammar consistently resulted in a high coverage on the test set (about 90%), with CL5 and CL10 ranked highest due to their feature variety. Grouping the models by accuracy, the best group is CL3 and CL4 and their lexicalised models (CL8 and CL9) with similar accuracy. This similarity implies the highly related features that these models are deploying; in other words, the presence of the parent node in the features has no added value to reranking. Comparison of accuracies over T_e and T for CL1-CL5 models with their corresponding lexicalised models (CL6-CL10) reveals that both accuracies follow the same trend: There is a consistent decline from CL1 to CL3, no change to CL4, followed by an increase at CL5 which its value is slightly smaller than CL2. As expected, the lexicalised models outperforms the unlexicalised models; but it is not a major gap. CL6 that only uses CS-PC has the highest contribution to reranking (highest accuracy over T_e and T).

Models with features from small-scale grammar (4.5 (b)) have a noticeable lower coverage. Inspecting the training and test set, many pairs happen to have an identical feature set for the preferred and dispreferred sample sentences which forces the system to back off to LM for such pairs. These frequent identical features are the outcome of using the features from right-branching structures with only two non-terminal symbols: FRAGMENTS, TOKEN. Accuracies for T_e are surprisingly better than the expectation with regards to the amount of information captured by the features. We will discuss this unexpected result in Section 4.3.3. The total

Model	T_e		acc.%
	cov%	acc.%	
FL1	64.66	63.96	66.38
FL2	54.17	69.6	69.52
FL3	45.66	54.8	62.6
FL4	67.58	55.03	60.15
FL5	64.75	64.81	67.07
FL6	45.4	57.29	63.71
FL7	66.04	69.01	69.73
FL8	69.3	69.17	69.43

(a)

Model	T_e		acc.%
	cov.%	acc.%	
FS1	61.56	84.08	79.49
FS2	67.41	91.2	86.11
FS3	0	0	70.42
FS4	63.2	54.15	60.83
FS5	63.11	79.16	76.66
FS6	0	0	70.42
FS7	61.39	84.31	79.66
FS8	71.45	87.24	83.96

(b)

Table 4.6: Results of reranking with f-structure features extracted from the large grammar (a), and the small grammar (b).

accuracy has remained acceptably high after the back-off to the baseline.

Using the small-scale grammar, the gap between the accuracy of lexicalised and unlexicalised models becomes larger. However, there is still an unexpected result: No improvement is achieved by substituting PoS tags in CS3-CS4 with the actual words in CS8-CS9. In other words there is no difference in the captured information by either of these features. Excluding CS8 and CS9, the total accuracies within each category (lexicalised and unlexicalised) do not vary significantly. The best model with small-scale grammar is CS10, the combination of all the c-structure features, with the highest accuracy over T_e and T .

If we were to compare results for the large-scale and small-scale grammars directly, it would seem that features from the small-scale grammar are better at deciding among alternative realisations. However, this is not a suitable comparison. We discuss the issue of grammar size further in Section 4.3.4.

F-structure only models Table 4.6 compares the reranking results for models with f-structure features using large-scale (Table 4.6(a)) and small-scale (Table 4.6(b)) grammar. These models and their corresponding features were previously summarised in Table 4.3.

On average the coverage of the models for the large-scale grammar is a bit lower than coverage for CS models (Table 4.5). Similar to the small-scale grammar case in c-structure models, there are many pairs with identical feature sets. This is due to the fact that f-structure describes the relationship between the grammatical components of a sentence regardless of its word order. Considering the fact that all these alternatives were built upon the same f-structure at the first stage, the probability of getting a very similar or identical f-structure is high.

Accuracy over T_e implies that f-structure features, unlike c-structure features, cannot effectively distinguish between the positive and negative alternatives in a pair. As a matter of fact these features are no better than the baseline for the large-scale grammar. So even though our task context differs a little from Cahill et al. (2007a), such that could potentially have been useful, it turns out that they are not.

Switching from large-scale grammar to small-scale grammar, the coverage for T_e has seen no significant change except for FS3 and FS6 that suddenly dropped to 0. These two models use FS-LEX-SUBCAT and FS-VERB-ARGS features, respectively. Regarding the above mentioned reason for identical feature sets in pairs, the very simplistic grammar and the nature of the features explains why the all the pairs ended up with identical feature set for both sentences.

The most striking result so far belongs to the accuracy over T_e using the small grammar. Considering the accuracies from large and small-scale for C models (CL vs. CS) along with the unexpectedly low accuracies for FL models lowered the accuracy expectation for the FS models to the same level or a bit higher; however, a sudden increase has been observed in the accuracies, except for FS4. This unexpected rise can be explained by the fact that FL models suffer from sparsity versus FS models, which have a very limited number of features and so their frequency for each alternative becomes a reliable distinguishing factor. FL7 and FS2 are the best reranking models with f-structure features for the large and small grammars respectively.

4.3.2.2 Combination of C- and F-structure Models

To answer the second research question for this chapter, we introduce multiple hybrid systems² that are incrementally built upon the models discussed so far. There are three sets of hybrid models: (1) Combination of the best c-structure model with the best f-structure model for small and large grammars (CFS and CFL), (2) Combination of the best c-structure model with various dependency models (CS+SD, CS+UD, CL+SD and CL+UD) and (3) Combination of the best c-/f-structure with dependency features (CFS+SD, CFS+UD, CFL+SD and CFL+UD).

These models are listed in Table 4.7. The first column lists the model names and the rest of the columns specify the other features/models which their features are deployed. The hybrid models follow the same self-descriptive convention as before, each letter in the name specifies the presence of its best model, e.g. **CFS** stands for the combination of features from **CS10** and **FS2** extracted from the Small-scale grammar. Since all the hybrid models with features from the small

²Each hybrid system feature combines features from specified base models with equal weight.

Model	parserType		LFG models		
	parser	Dep. Models	c-structure	f-structure	grammar
CFS	-	-	CS10	FS2	small
CFL	-	-	CL6	FL7	large
CS+SD[1-14]	Sup	D[1-14]	CS10	-	small
CS+UD[1-14]	Unsup	D[1-14]	CS10	-	small
CL+SD[1-14]	Sup	D[1-14]	CL6	-	large
CL+UD[1-14]	Unsup	D[1-14]	CL6	-	large
CFS+SD[1-14]	Sup	D[1-14]	CS10	FS2	small
CFS+UD[1-14]	Unsup	D[1-14]	CS10	FS2	small
CFL+SD[1-14]	Sup	D[1-14]	CL6	FL7	large
CFL+UD[1-14]	Unsup	D[1-14]	CL6	FL7	large

Table 4.7: Naming convention for various models using dependency structure

model	T_e		acc.%
	cov%	acc.%	
CFS	69.91	94.83	76.43
CFL	88.22	84.75	77.37

Table 4.8: Reranking results for the combination of the best c-structure and f-structure models

scale grammar would have CS10 and/or FS2 prefix, the names are contracted to letters only, followed by the grammar size, CS/CFS. The same argument is valid for models with large scale grammar features, CL and CFL are the abbreviated forms of CL6 and FL7, respectively.

CF models The first two combinations investigate the quality of reranking by combining the best c- and f-structure models. The results in Table 4.8 present the coverage and accuracy for CFS and CFL. Along the lines of the previous results, the same trade-off exists between the coverage and accuracy for small and large grammar. The accuracies become quite close after backing off to the baseline which is LM.

c-structure + dependency features Table 4.9 reports the reranking results for hybrid models that combine dependency features with the best small-scale grammar c-structure feature. The main contribution of adding dependency parser features can be seen in the coverage over T_e ; about 26% increase from 69.99% (CS10 coverage) to 97.85% and 96.22% (the average coverage for CS+SD and CS+UD models). This dramatic change is due to the variety of these additional features that

Model	T_e			Model	T_e		
	cov%	acc.%	acc.%		cov.%	acc.%	acc.%
CS+SD1	96.5	86.64	86.17	CS+UD1	95.5	86.15	85.83
CS+SD2	96.6	86.88	86.42	CS+UD2	96.1	85.5	85.07
CS+SD3	98.3	85.71	85.48	CS+UD3	94.2	87.49	87.05
CS+SD4	98.4	84.87	84.6	CS+UD4	94.2	86.5	86.12
CS+SD5	96.8	88.04	87.57	CS+UD5	96.2	85.04	84.67
CS+SD6	98.5	86.4	86.13	CS+UD6	96.5	85.37	84.94
CS+SD7	98.5	86.76	86.49	CS+UD7	96.5	85.94	85.49
CS+SD8	97.1	89.97	89.45	CS+UD8	97.3	87.17	86.74
CS+SD9	96.9	85.34	85.01	CS+UD9	97.4	84.64	84.31
CS+SD10	98.7	83.95	83.64	CS+UD10	95.1	85.55	85.23
CS+SD11	98.7	83.18	82.88	CS+UD11	95.4	85.44	85.13
CS+SD12	97.1	89.83	89.31	CS+UD12	97.4	87.14	86.74
CS+SD13	99	87.91	87.73	CS+UD13	97.7	87.03	86.64
CS+SD14	99	88.31	88.13	CS+UD14	97.7	87.21	86.82

(a)

(b)

Table 4.9: Results of reranking using various combinations of (a) supervised and (b) unsupervised dependency parser features added to CS-structure features extracted from the small grammar

prevent identical feature sets in a pair. Comparing the coverage of the supervised versus unsupervised models, the numbers are quite close with the supervised parser almost always beating the unsupervised except for 4 models (CS+SD3/CS+SD10 and CS+SD4/CS+SD11). These dependency models deploy CS-SB (unlx/lx) and CS-NC (unlx/lx) features. The same relationship exists for the both accuracies: supervised models beat the unsupervised ones except for the four aforementioned models. In the previous chapter, we had to do feature selection to get a close result with unsupervised parser and make it comparable to supervised parser. However, in the hybrid models, the accuracies for these two features types are quite close (on average less than 2%) which suggests that c-structure features dominates the decision making. Comparing all the overall accuracies from Tables 4.5, 4.6, 4.8, and 4.9, CS10 in Table 4.5 still leads with a minor superiority. It means there is no added information in the dependency features to boost the reranking quality.

As can be seen in Table 4.10, switching to the large-scale grammar for c-structure features, the same trend for coverage is still valid. Similarly, the models with supervised features slightly outperformed the unsupervised ones, except for CL+SD10 and CL+SD11. Similar to CS+SD and CS+UD models, the overall accuracies are better than the statistical parser models only but none of the CL+SD or CL+UD models beat CL6.

Model	T_e			Model	T_e		
	cov%	acc.%	acc.%		cov.%	acc.%	acc.%
CL+SD1	96.9	84.25	83.92	CL+UD1	98.1	83.83	83.62
CL+SD2	96.9	85.17	84.82	CL+UD2	98.2	83.39	83.22
CL+SD3	97.4	85.1	84.76	CL+UD3	96.7	84.5	84.15
CL+SD4	97.5	84.99	84.68	CL+UD4	96.7	83.77	83.4
CL+SD5	97	85.02	84.65	CL+UD5	98.2	83.79	83.6
CL+SD6	97.5	84.87	84.52	CL+UD6	98.5	83.47	83.25
CL+SD7	97.5	84.68	84.33	CL+UD7	98.5	83.92	83.69
CL+SD8	97	85.04	84.71	CL+UD8	99.1	83.99	83.92
CL+SD9	96.9	84.36	84.06	CL+UD9	99.1	82.32	82.23
CL+SD10	97.8	83.3	83.03	CL+UD10	97	83.78	83.49
CL+SD11	97.8	83.36	83.12	CL+UD11	97.1	83.49	83.19
CL+SD12	97	85.13	84.8	CL+UD12	99.1	84.16	84.09
CL+SD13	97.9	84.38	84.07	CL+UD13	99.3	84.1	84.04
CL+SD14	97.9	84.38	84.07	CL+UD14	99.3	84.01	83.95

(a)

(b)

Table 4.10: Results of reranking using various combinations of (a) supervised and (b) unsupervised dependency parser features added to CL-structure features extracted from the large grammar

c- and f-structure + dependency features The final two experiments involve adding f-structure features to the CS+SD, CS+UD, CL+SD and CL+UD models.

In Table 4.11, the results for CFS+SD can be compared with CFS (Table 4.8) and CS+SD (Table 4.9). CFS+SD8 is the best model amongst the hybrids of supervised features, with accuracy 88.82. This model is ranked lower than CS+SD8 (89.45), but higher than CFS (76.43). Switching to dependencies from the unsupervised parser, CFS+UD8 is the best model amongst all the CFS+UD models. As expected its accuracy, 86.60, is less than its supervised peer but maintain the same ranking position: below CS+UD3(87.05) but above CFS(76.43). The coverage over the effective set has been slightly improved due to the addition of f-structure features.

Similarly, the CFL+SD results (Table 4.12) can be compared to the three models which its features are incrementally derived from: CL+SD (Table 4.10) and CFL (Table 4.8). For hybrid models with supervised dependency features, CFL+SD7 with accuracy 84.03 is ranked above CFL, 77.37, but below CL+SD2 84.82. Switching to unsupervised features, CFL+UD8 with accuracy 83.84, is slightly lower than its supervised peer, ranked above CFL but below its CL+UD3 unsupervised peer (84.15).

These two experimental results reinforce the initial assumption that no extra rerank-

Model	T_e			Model	T_e		
	cov%	acc.%	acc.%		cov.%	acc.%	acc.%
CFS+SD1	96.5	87.11	86.62	CFS+UD1	95.6	85.8	85.48
CFS+SD2	96.6	87.54	87.05	CFS+UD2	96.2	85.62	85.22
CFS+SD3	98.3	84.9	84.68	CFS+UD3	94.3	86.62	86.22
CFS+SD4	98.4	84.24	83.98	CFS+UD4	94.3	86.23	85.85
CFS+SD5	96.8	88.22	87.74	CFS+UD5	96.3	85.26	84.91
CFS+SD6	98.5	86.13	85.87	CFS+UD6	96.6	85.59	85.18
CFS+SD7	98.5	87.3	87.02	CFS+UD7	96.6	85.96	85.54
CFS+SD8	97.1	89.32	88.82	CFS+UD8	97.4	87	86.6
CFS+SD9	96.9	85.15	84.83	CFS+UD9	97.5	84.3	84.01
CFS+SD10	98.7	83.95	83.64	CFS+UD10	95.2	85	84.73
CFS+SD11	98.7	83.09	82.79	CFS+UD11	95.4	84.99	84.69
CFS+SD12	97.1	88.9	88.41	CFS+UD12	97.5	86.8	86.4
CFS+SD13	99	87.47	87.3	CFS+UD13	97.8	86.6	86.25
CFS+SD14	99	87.51	87.34	CFS+UD14	97.8	86.78	86.43

(a)

(b)

Table 4.11: Results of reranking using various combinations of (a) supervised and (b) unsupervised dependency parser features added to c-/f-structure features extracted from the small grammar

ing helpful information can be extracted from f-structure features in reranking (Cahill et al., 2007a).

To summarise the results for reranking with dependency models combined with internal structure features, Table 4.13 lists the best models from each dependency hybrid model, grouped by grammar size in an ascending order. The main result of the chapter is that the best of the external features didn't improve over just internal features. Comparing models with external features from supervised vs. unsupervised parsers, the hybrid models rank similarly across both grammars: regardless of the parser type, CS+D has been ranked the highest, followed by CF+SD in the second place and finally, CL+D and CFL+D have been ranked as the third and fourth hybrid models.

4.3.3 Discussion on the Results of the Small Grammar

The simple grammar consistently produced high accuracies over a smaller T_e compared to the large grammar which seemed to be unexpected since the small grammar produces the same right-branching tree for every input sentence: feature sparsity would no longer be a problem and due to the repetition of the TOKEN and FRAGMENT non-terminals, features that include these two labels would get a

Model	T_e			Model	T_e		
	cov%	acc.%	acc.%		cov.%	acc.%	acc.%
CFL+SD1	97	84.19	83.85	CFL+UD1	98.2	82.76	82.61
CFL+SD2	96.9	83.8	83.52	CFL+UD2	98.2	82.7	82.53
CFL+SD3	97.5	83.93	83.65	CFL+UD3	96.9	83.5	83.19
CFL+SD4	97.5	83.26	82.95	CFL+UD4	96.9	83.19	82.93
CFL+SD5	97.1	83.75	83.41	CFL+UD5	98.3	82.99	82.77
CFL+SD6	97.5	83.78	83.49	CFL+UD6	98.5	82.58	82.32
CFL+SD7	97.5	84.33	84.03	CFL+UD7	98.5	82.31	82.05
CFL+SD8	97.2	84.11	83.79	CFL+UD8	99.1	83.5	83.43
CFL+SD9	97.1	83.38	83.05	CFL+UD9	99.1	81.94	81.89
CFL+SD10	97.8	82.34	82.09	CFL+UD10	97.2	83.32	83.02
CFL+SD11	97.8	81.99	81.78	CFL+UD11	97.2	83.12	82.81
CFL+SD12	97.2	84.11	83.79	CFL+UD12	99.1	83.32	83.26
CFL+SD13	97.9	83.58	83.32	CFL+UD13	99.3	83.17	83.11
CFL+SD14	97.9	83.49	83.24	CFL+UD14	99.3	83.35	83.29

(a)

(b)

Table 4.12: Results of reranking using various combinations of (a) supervised and (b) unsupervised dependency parser features added to c-/f-structure features extracted from the large grammar

Model	acc.%	Model	acc.%
CS10	89.81	CL6	85.77
CS+SD8	89.45	CL+SD2	84.82
CFS+SD8	88.82	CFL+SD7	84.3
CS+UD3	87.5	CL+UD3	84.15
CFS+UD8	86.6	CFL+UD8	83.43
FS2	86.11	CFL	77.37
CFS	76.43	FL7	69.73

(a)

(b)

Table 4.13: Summary of ranking results for various combinations of features from external resources with (a) the small grammar and (b) the large grammar. Regardless of the size of the grammar, and the best of the external features didn't improve over just internal.

higher occurrence rate per sentence. For instance for the c-structure presented in Figure 4.9, only 14 features per each CS-PC and CS-GPC can be extracted (see Table 4.14). Two of these features consist of non-terminals with a high frequency and the rest of the features consist of a non-terminal and a terminal node with only one occurrence. A similar argument is valid for features that represent the horizontal slices of the c-structure. Since such features cannot capture any sentence-

Encoded Feature	Feature	#Occurence
F_CS_1	FRAGMENTS-TOKEN	13
F_CS_2	FRAGMENTS-FRAGMENTS	13
F_CS_3	TOKEN-Pierre	1
F_CS_4	TOKEN-Vinken	1
F_CS_5	TOKEN-,	2
⋮	⋮	⋮
F_CS_14	TOKEN-.	1
F_GP_1	FRAGMENTS-FRAGMENTS-TOKEN	12
F_GP_2	FRAGMENTS-FRAGMENTS-FRAGMENTS	12
F_GP_3	FRAGMENTS-TOKEN-Pierre	1
F_GP_4	FRAGMENTS-TOKEN-Vinken	1
F_GP_5	FRAGMENTS-TOKEN-,	2
⋮	⋮	⋮
F_GP_14	FRAGMENST-TOKEN-.	1
F_SB_1	TOKEN-FRAFMENTS	12
F_SB_2	Pierre	1
F_SB_3	Vinken	1
F_SB_4	,	2
⋮	⋮	⋮
F_SB_13	.	1

Table 4.14: Encoded CS-PC and CS-GPC features along with their count. F_CS_3–F_CS_14 and F_GP_3–F_GP_14 are following are following FRAGMENTS- w_i and FRAGMENTS-FRAGMENTS- w_i templates, where w_i can be replaced by any word from sentence (2.1). As discussed above, the small grammar is unable to assign a detailed structure including grouping words into phrases and so on. Consequently, the features in such a shallow tree becomes quite repetitive, e.g., FRAGMENTS-TOKEN has been observed 13 times

specific structure, all the sentences will get the TOKEN-FRAGMENTS slice in addition to the unigrams of their terminals. This implies none of the c-structure features are able to encode the word order into the model.

The number of unique features will be dropped even further for unlexicalised features. The unlexicalised version of F_CS_1 and F_CS_2 would remain unchanged but the 12 unique TOKEN-*word* features would be reduced to 7: FRAGMENTS-TOKEN-*PoS*, where *PoS* may refer to the PoS tag of any given word in the sentence: $Pos \in \{N, COMMA, A, V, P, D, NUMBER, .\}$.

Features presented in Table 4.14 can also confirm that none of the features capture any word order-relevant information. To understand this curious effect, we had to look into the data set. To recap, we had deployed the large grammar to produce alternatives and we followed the random pairing (see page 56) to pair alternatives. Remember that the small grammar is capable of regenerating more than one alternative which is a match for the input sentence. The generated alternatives for sentence (2.1) have been labeled with A_1 to A_4 as follows:

A_1 : Pierre Vinken, former Elsevier chairman died at the age of 83.

A_2 : Pierre Vinken, former Elsevier chairman, died at the age of 83.

A_3 : At the age of 83 pierre Vinken, former Elsevier chairman died.

A_4 : At the age of 83 pierre Vinken, former Elsevier chairman, died.

These alternatives vary in both length and word order, but due to the nature of the small grammar, the features describing these alternatives are only capable of reflecting sentence length and word form — in lexicalised models — but not word order differences. For the very same reason, only four pairs, $\{\{A_1, A_2\}, \{A_2, A_3\}, \{A_3, A_4\}, \{A_1, A_4\}\}$, out of 6 possible pairings, would get non-identical feature set for preferred and dispreferred instances, as they differ in length: one instance has an extra comma. Depending on the alternatives, the feature set may differentiate by containing a new feature (in the case of word form differences) or an increment to the number of occurrence of the feature (additional word/punctuation).

This explains why approximately 30% (depending on the model) of the test set has not been covered in the experiment. The further drop in the coverage for the models with the unlexicalised features can be explained by the fact that sometimes alternatives only differ in the word form: e.g. U.S. vs. US which after de-lexicalisation features sets become identical.

As a result we can conclude that with the small grammar, the model indeed become sensitive to unigram surface features such as punctuation and word forms since other structural features are almost the same for both preferred and dispreferred sentences.

4.3.4 Summary

The experiment results support the usefulness of text generator internal structures as a feature source for reranking, providing the suitable grammar and feature types.

Combining all the lexicalised c-structure feature types extracted from the small grammar has affected the reranking quality the best. On the other hand, F-structure features extracted from the large grammar (FL) have the least contribution to reranking: the accuracies are even worse than baseline (just using language model).

Similar to German (Cahill et al., 2007a), the symbolic grammar’s internal structure contributes to the reranking of the generated text in English. This finding suggests that in practice such features can be used universally and independent from the language or formalism deployed by the realiser.

However, the answer to our second question for this chapter, RQ4.2 on page [91](#), is negative. Adding in external structural features to the internal ones does not do better than internal ones alone, suggesting that the structures are too similar rather than being usefully complementary.

Chapter 5

An ILP Framework for String Regeneration

5.1 Introduction

In this chapter and the next, we turn to the second task that we tackle in this thesis. String regeneration or general purpose word ordering refers to the task of assigning word order to a given bag-of-words. This task can be regarded as a subtask in a number of Natural Language Generation (NLG)-related applications, such as Text Summarisation (Wan et al., 2009) and Text Generation (Reiter and Dale, 1997) itself.

In Chapter 1, we discussed how the linearisation task in NLG can be regarded as the reverse of parsing, in the sense of transition between a sentence and its grammatical structure form where the former task’s search space can be exponentially larger. In Section 2.3.2, we looked at one the popular approaches for surface realisation: finding an intermediate structure for a given bag-of-words and then linearising it. Several of these recent approaches share a common ground: a machine learning method that learns features’ weights; a search algorithm that imposes a structure on the input e.g., a parse tree; and a lineariser which flattens the structure into a sentence. Structures can be as simple as assigning n-grams (Langkilde and Knight) or as intricate as using Latent Semantic Analysis (LSA) to model the language (Dennis, 2013).

Zhang (2013) characterises string regeneration from bag-of-words as one end of a cline, with linearisation from a full syntactic structure at the other; the cline corresponds to the availability of varying amounts of input structure. For instance, take sentence (5.1) as the target string for the string regeneration problem. On one end of the cline, the input has been annotated with minimal structure: bag-of-words along with Part of Speech (PoS) tags (see the set of words in (5.2)). In the

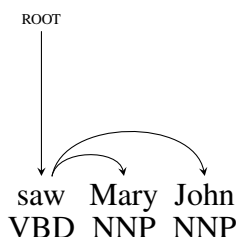


Figure 5.1: The bag-of-words (5.2) annotated with gold standard dependencies that can be served as an input to the ordering task.

middle of the cline, some structural information has been incorporated in the input; dependency relations between tokens constitute one such piece of information. Figure 5.1 contains dependencies for the bag-of-words in (5.2).

(5.1) John saw Mary.

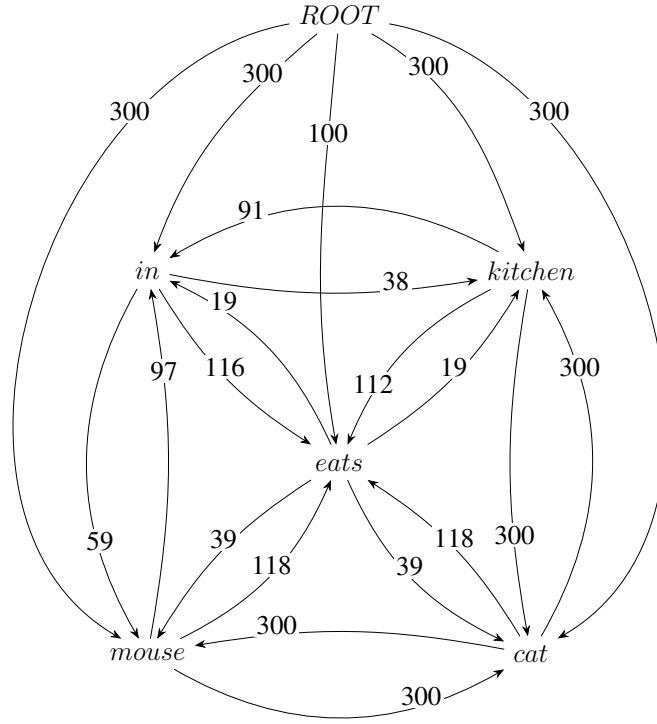
(5.2) {saw/VBD, Mary/NNP, John/NNP}

We are interested in approaches to string regeneration that consider assigning a syntactic form to the input as an intermediate task to regeneration, since it can benefit from any existing structure that comes with the input by directly embedding it into that intermediate structure and this can help the regeneration. We discussed some previous works that follow this approach in Section 2.3: to recap, Wan et al. (2009) took this approach and chose dependency trees as the intermediate syntactic form. They represented the bag-of-words with a complete weighted digraph so that edge $e_{i,j}$ serves as a potential dependency between words w_i and w_j — see Figure 5.2(a) that represents such a graph for Sentence (5.3).

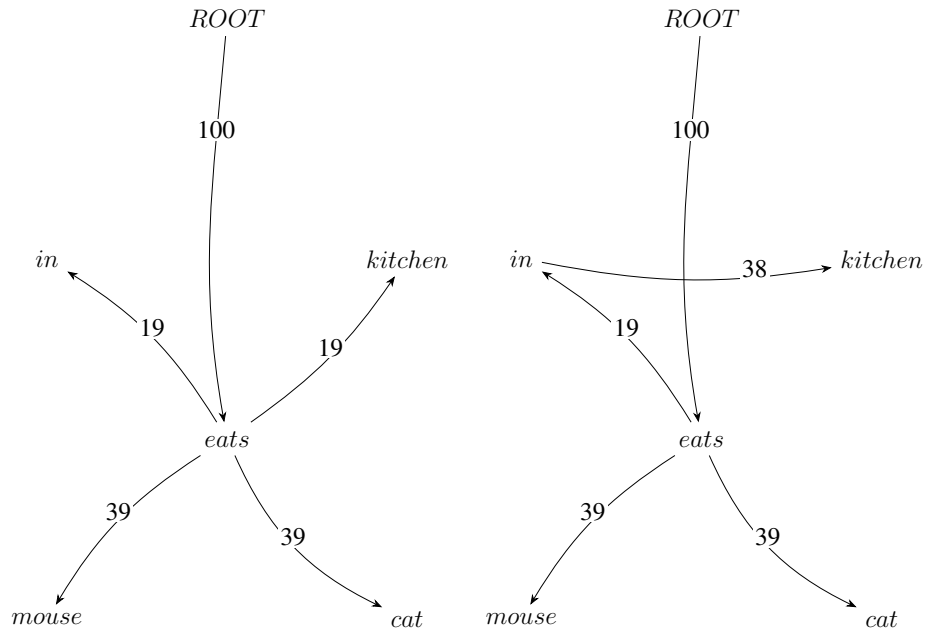
(5.3) the cat eats the mouse in the kitchen,

They extracted dependency weights from the Penn Treebank, by calculating the log probability of such a dependency when these two words were observed in one sentence. Their aim was to construct a dependency tree by extracting a Minimum Spanning Tree (MST) from the weighted digraph. They applied the CLE algorithm (see Section 2.3.2) to get the MST. Since the weights are calculated using a dependency treebank, the MST can be viewed as a dependency tree.

Since the CLE algorithm is a pure graph algorithm, its only criteria for selecting an edge to participate in the final MST is that edge’s weight which is typically a function of co-occurrence of the head and dependent. For instance compare the two possible dependencies between *eats* and *kitchen* in Figure 5.2(a): $eats \xleftarrow{112} kitchen$ shows a weaker association than its counterpart $eats \xrightarrow{19} kitchen$ dependency, i.e., *eats* is more likely to be the head of *kitchen*. The same argument is valid



(a) The complete graph for sentence (5.3)



(b) CLE-generated MST

(c) ILP-generated MST

Figure 5.2: Comparison of the two MSTs generated by Wan et al. CLE-based approach and ILP for Sentence (5.3). Note that base NPs are represented by their head. e.g. *cat* is the head of *the cat* base NP.

for $in \xleftarrow{91} kitchen$ and $in \xrightarrow{38} kitchen$ dependencies where the latter forms a stronger association. This means *in* being the parent of *kitchen* is a more plausible choice than the other way around. So, *in* and *eats* can be the potential heads for *kitchen*. The CLE algorithm chooses $eats \rightarrow kitchen$ as it has a larger weight (Figure 5.2(b)).

This choice, however is not a top pick from a linguistic perspective as it makes both *eats* and *in* suffer from an incorrect number of arguments. To address this issue and allow linguistically plausible structures, Wan et al. (2009) additionally drew on the graph-theoretic assignment problem as a way of matching heads and dependents and deployed the Hungarian algorithm. With this algorithm they managed to have more control over each node’s arity by assigning a maximum number of dependents to different word categories.

Wan et al. do not incorporate existing information into the input which is later addressed by Zhang and Clark (2011). They featured the use of input from the cline ranging from PoS tags to subtree in their bottom-up approach of building a Combinatory Categorical Grammar (CCG) parse tree as the intermediate syntactic representation. Clearly, the generated string would be closer to the target sentence as the input gets closer to the richer end of the cline.

Another shortcoming of the above mentioned frameworks is that adding or modifying dependency tree properties is an elaborate task, as they are implicitly embedded into the model e.g., in the form of weights or hypothesis scores. It would be desirable to be able to declare structural requirements along with its linguistic properties independently and impose them on the problem at once or incrementally.

In this part of the thesis we propose a string regeneration framework which is graph-based (similar to Wan et al.) and is able to consume input from the Zhang and Clark’s cline. Additionally, this framework would allow the declarative incorporation of the structural and linguistically-motivated constraints. ILP seems to be a promising choice since it allows a wide variety of constraints to be imposed to the problem at once or incrementally. To date no research has employed ILP to create a dependency tree from a bag-of-words but given its contribution to dependency parsing (Riedel and Clarke, 2006, Martins et al., 2009) and Wan et al.’s successful adaptation of McDonald et al.’s dependency parsing for string regeneration suggests the usefulness of introducing ILP to our framework.

We aim for a string regeneration framework that can benefit from any available structural information without having a set expectation of the input richness. Such structural information will serve as the basis of our proposed framework, rather than a complementary piece of information that can contribute to the quality of the output. This means the estimation of missing structures will be the first step in our regeneration process. This makes the representation of such information in the framework of a high importance as each structure (1) must be encoded

independently of the others i.e., to provide the aforementioned flexibility (2) must include information about word order within its own scope, and (3) must have feature(s) that allow data-driven approaches to performing modelling from the existing structures and estimating the missing ones.

Such a framework will suit languages which lack linguistic resources to induce structural information as well as languages that are in the process of developing resources and have little structural information to start with (see discussion on p.8). It is worth re-emphasising that the languages we are interested in can use universal dependency parsers to cover resources used in this part of the research. For instance, a parser can be built based on a universal parser or transfer-based approach discussed earlier in Chapter 3. This parser can be used to build a treebank to get the dependency weights and branching factors.

After a quick review of ILP and its application to dependency parsing, where we give the intuition behind how ILP can be applied to our generation problem, we will discuss the technical aspects of our proposed framework.

5.2 Related work

5.2.1 Brief Review of ILP

Translation of the real-world optimisation problem into a mathematical form: an objective function and a set of constraints is a standard method for solving optimisation problems. ILP is a mathematical model that optimises an objective function subject to some linear constraints. Constraints can be described with either equalities or inequalities or both. A full overview is available from a standard text, such as Nemhauser and Wolsey (1988); we are just going to give a brief review below. The canonical form of an ILP problem as defined by Nemhauser and Wolsey (1988) is:

$$\begin{aligned}
 &\text{maximize:} && F(x) = \mathbf{c}^T x \\
 &\text{subject to:} && \mathbf{A}x \leq \mathbf{b}, \\
 &&& x \geq 0, \text{ and} \\
 &&& x \in \mathbb{Z}
 \end{aligned} \tag{5.1}$$

In the definition (5.1), the objective function and all the constraints have to be linear and all the variables must be integer. \mathbf{c} and \mathbf{b} are integer vectors while \mathbf{A} is a matrix with integer values. Related methods follow the same canonical form but

differ in details: for example, for quadratic programming the objective function is not a linear function any more.

One of the standard methods to solve ILP problems is the graphical method. The first step is to construct a graphical representation of the feasible region by plotting the constraints and identifying the area of the valid side of the constraints. The second step is to draw two objective function lines to determine the direction of improvement. The next step is finding the optimal point by moving the objective function line along the desired direction, and algebraically calculating the coordinates of the optimal point. The final step is the calculation of the objective function value for the optimal solution.

Consider an example problem from Bertsimas and Tsitsiklis (1997):

$$\begin{aligned} \text{minimise:} \quad & -x_1 - x_2 \\ \text{subject to:} \quad & 2x_1 + 2x_2 \leq 3, \\ & x_1 + 2x_2 < 3 \text{ and} \\ & x_1, x_2 > 0 \end{aligned} \tag{5.2}$$

This optimisation problem can be regarded as a cost minimisation problem for manufacturing two products: x_1 and x_2 and each constraint describes one of the limitations that the producer has to deal with, for instance, the availability of the raw material and the expected labour, respectively.

The first step is identifying the feasible region using the provided constraints — the shaded area in Figure 5.3. To find an optimal solution, we follow the proposed steps above. For any given scalar z , we consider the set of all points whose cost $c'x$ is equal to z ; this is the line described by the equation $-x_1 - x_2 = z$. Note that this line is perpendicular to the vector $c = (-1, -1)$. Different values of z lead to different lines, all of them parallel to each other. In particular, increasing z corresponds to moving the line $z = -x_1 - x_2$ along the direction of the vector c . Since we are interested in minimising z , we would like to move the line as much as possible in the direction of $-c$, as long as we do not leave the feasible region. The best we can do is $z = -2$ (see Figure 5.3), and the vector $x = (1, 1)$ is an optimal solution.

However, not all optimisation problems are as simple to formulate. Operations Research (OR) is one of the fields that employs optimisation methods to model real life problems such as production planning, scheduling, telecommunication networks, etc. These tasks are crucial to businesses as they can directly affect the business and the quality of service/product by optimising the processes and potentially offer some cost cutting. An example is the fleet assignment problem in the airline industry. For instance according to Delta, after applying the optimised schedule, the savings during the period from June 1 to August 31, 1993 were estimated at about \$220,000 per day over the old schedule. Rajgopal (2004)

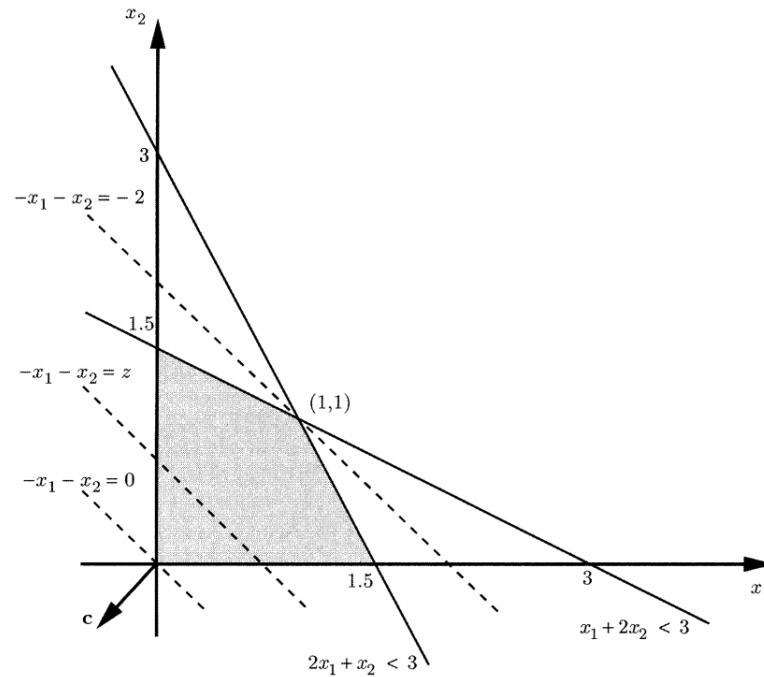


Figure 5.3: Graphical solution of the problem in Example (5.2), reproduced from Bertsimas and Tsitsiklis (1997).

described the problem as follows:

Delta Airlines flies over 2500 domestic flight legs each day and uses about 450 aircraft from 10 different fleets, and the objective was to assign aircraft to flight legs in such a way that revenues from seats are maximised. The tradeoff is quite simple — if a plane is too small then the airline loses potential revenue from passengers who cannot be accommodated on board, and if it is too large then the unoccupied seats represent lost revenue (in addition to the fact that larger aircraft are also more expensive to operate). Thus the objective is to ensure that an aircraft of the “correct” size be available when required and where required. Unfortunately, ensuring that this can happen is tremendously complicated since there are a number of logistical issues that constrain the availability of aircraft at different times and locations.

Subramanian et al. (1994) modelled this problem using a very large mixed-integer¹ linear program — a typical formulation could result in about 60,000 variables and 40,000 constraints. The planning horizon for each problem is one day since the assumption is made that the same schedule is repeated each day².

¹Only some of the variables, x_i , are constrained to be integers, while other variables are allowed to be non-integers.

²Exceptions such as weekend schedules are handled separately.

objective function tries to minimise the sum of operating costs (including such things as crew cost, fuel cost and landing fees) and costs from lost passenger revenues.

constraints fall into one of the following groups: (1) conservation of flow of aircraft from the different fleets to different locations around the system at different scheduled arrival and departure times; (2) governing the assignment of specific fleets to specific legs in the flight schedule; and (3) the availability of aircraft in the different fleets, regulations governing crew assignments, scheduled maintenance requirements, and airport restrictions.

Traditionally, the first step in solving an ILP problem is to solve the relaxed problem, i.e, without any integrality constraints. Then, the next step is to work out the solutions that satisfy the constraints. ILP problems are known to be of NP-hard complexity in the worst case scenario. Depending on the complexity of the objective function, sparsity and dimensions of constraints matrix and the search method it can be solved in polynomial time. For instance, Branch and Bound is the generic search algorithm for the optimisation problem but if the problem is an intractable one, heuristic methods such as Simulated annealing, Hill climbing, etc suit better.

Models that describe real-world problems, such as the one discussed above, are generally large-scale and computationally intensive. That explains why many commercial and open-source software packages such as Gurobi³, LpSolve⁴, Symphony⁵, Clp⁶, GLPK⁷ provide highly sophisticated implementations of linear programming solvers and even APIs for various programming languages to allow effective modelling and solving of optimisation problems. We will discuss optimisation problems in the context of multi-objective optimisation later in Section 5.6.2.1.

Given the solid mathematical background of these methods, the availability of high quality computational packages, and most important of all their flexibility and scalability, we think of them as a suitable candidate to model NLP/NLG problems. In the next section we review some previous works that use ILP to address parsing and that inspired us to use this optimisation method to address the problem of generation from bag-of-words.

³<http://www.gurobi.com/>

⁴<http://lpsolve.sourceforge.net/5.5/>

⁵<https://projects.coin-or.org/SYMPHONY>

⁶<https://projects.coin-or.org/Clp>

⁷<https://www.gnu.org/software/glpk/>

5.2.2 ILP in Graph-based Parsing

We described graph-based dependency parsing, as introduced by McDonald et al. (2005b), in Section 2.3.1. Riedel and Clarke (2006) reformulated this graph-based labelled dependency parsing as an ILP problem, using constraints to define the MST. Under this approach, global constraints can be added to a problem individually as opposed to algorithms like CLE that have been applied to dependency parsing (as noted in Section 2.3.1), which lack this flexibility.

Unlike previous works that tend to solve the ILP problem at one go, Riedel and Clarke (2006) considered the incremental addition of constraints to improve efficiency and also to avoid intractable ILP problem when searching for a labelled dependency parse tree. They modified the underlying dependency model of McDonald et al. (2005b) to include dependency labels. Their function to optimise the total weight of the dependency tree as the sum of the edge weight is:

$$s(\mathbf{x}, \mathbf{y}) = \sum_{(i,j,l) \in \mathbf{y}} s(i, j, l) \quad (5.3)$$

$$= \sum_{(i,j,l) \in \mathbf{y}} \mathbf{w} \cdot \mathbf{f}(i, j, l) \quad (5.4)$$

where, \mathbf{x} is a sentence, \mathbf{y} is a set of labeled dependencies, $\mathbf{f}(i, j, l)$ is a multi-dimensional feature vector representation of the edge that connects token i to token j with label l with weight \mathbf{w} . Finding the best parse tree amounts to finding the \mathbf{y} for a given \mathbf{x} that maximises $s(\mathbf{x}, \mathbf{y})$:

$$\mathbf{y}' = \operatorname{argmax}_{\mathbf{y}} s(\mathbf{x}, \mathbf{y})$$

They extended this basic model to conform with ILP so that each labelled edge is represented with a binary variable:

$$e_{i,j,l} \quad \forall i \in 0..n, j \in 1..n, l \in \text{best}_k(i, j)$$

where index 0 specifies root of the dependency tree, n is the number of tokens, and $\text{best}_k(i, j)$ is the set k labels with the highest $s(i, j, l)$. $e_{i,j,l}$ is 1 if the specified edge exists in the parse, otherwise 0. $\text{best}_k(i, j)$ provides the set of top k labels for the dependency that exists between token i and j . Additionally, they used a set of binary auxiliary variables $d_{i,j}$, $\forall i \in 0..n, j \in 1..n$ to represent the existence of a dependency between tokens i and j . They connected these to the $e_{i,j,l}$ variables by

the constraint:

$$d_{i,j} = \sum_{l \in \text{best}_k(i,j)} e_{i,j,l}$$

In Riedel and Clarke’s ILP framework, the objective function is then

$$\sum_{i,j} \sum_{l \in \text{best}_k(i,j)} s(i,j,l) \cdot e_{i,j,l}$$

which picks the best parse tree, subject to (1) tree constraints and (2) linguistically motivated constraints that altogether ensure a well-formed tree.

The “Base Constraints” as their name implies are added to the problem at the start. For instance ONLY-ONE-HEAD is a tree constraint and ensures each word has one and only one head, and is defined as:

$$\sum_i d_{i,j} = 1$$

for non-root tokens ($j > 0$) in \mathbf{x} , with an exception for an artificial root node, where ($j = 0$):

$$\sum_i d_{i,0} = 0$$

UNIQUE-LABEL also belongs to “Base Constraints” and is a linguistically motivated constraint that ensures each head word with index i has at most one outgoing edge with label l :

$$\sum_j e_{i,j,l} \leq 1$$

On the other hand, the “Incremental Constraints” are those constraints that are too expensive to be added in advance. Hence they are added to solve the problem only in the cases where they are violated. For instance NO-CYCLE, which forbids cycles in the tree, consists of multiple constraints depending on the tree size (see Section 5.5). Enumerating all the possible cycles and adding them to the ILP problem creates a bottleneck in parsing. However, not many of the candidate dependency structures produced by base constraints are cyclic. Therefore, it is plausible to create the dependency parse tree first and if it is cyclic, re-solve the ILP problem with a specific constraint that forbids the cycle. This process continues until an acyclic structure is produced as MST.

For speed reasons, Riedel and Clarke (2006) trained their dependency model using the CLE search algorithm and single-best Margin Infused Relaxed Algorithm

(MIRA) as a learner on the Dutch Alpino Corpus. Then this model uses ILP at test time to study how effectively it can recognise appropriate dependencies. Comparing with baseline, the incorporation of the linguistic constraints resulted in performance increases of 0.5% for both labelled and unlabelled accuracy.

Later, Martins et al. (2009) introduced a concise ILP formulation for non-projective dependency parsing with a polynomial size. Their formulation improved on Riedel and Clarke (2006)'s work in three different areas: (1) The number of variables and constraints are polynomial in the sentence length, therefore no need for incremental procedure, (2) LP relaxation allows fast online discriminative training of the constrained model, and (3) there is the capability of learning soft constraints to handle higher-order arc interactions and model word valency. This model accepts prior knowledge as hard constraints but it is also able to learn soft constraints from data such as correlations among neighbouring arcs (siblings and grandparents), word valency, and tendencies toward nearly projective parses. They used a max-margin framework for learning the model's parameters using a linear programming relaxation.

They evaluate their model with multiple languages (English, Danish, Dutch, Portuguese, Slovene, Swedish and Turkish) using the unlabelled attachment score (UAS). This score represents the percentage of tokens with correct head. The reported performance showed improvement compared the existing state-of-the-art methods.

In the following section, we will discuss how we can adapt the successful use of ILP in parsing to the text generation problem.

5.3 An ILP Framework for Surface Realisation

This section describes the application of the ILP framework, introduced in Section 5.2.1, to the text generation problem. It focuses on the definition of the objective function and how the combination of linguistic constraints with tree constraints are responsible for creating plausible dependency trees.

The parsing work of McDonald et al. (2005b) described in Section 2.3.2 used a simple directed graph, where there is at most one edge from one vertex to another. The complication for text generation over parsing is that in addition to selecting which dependencies to include, we need to specify word order. Having no predefined word order makes four possible dependencies given a bag-of-words of size 2 (See Figure 5.4).

Consequently, we need to redefine the dependency notation to include relative word

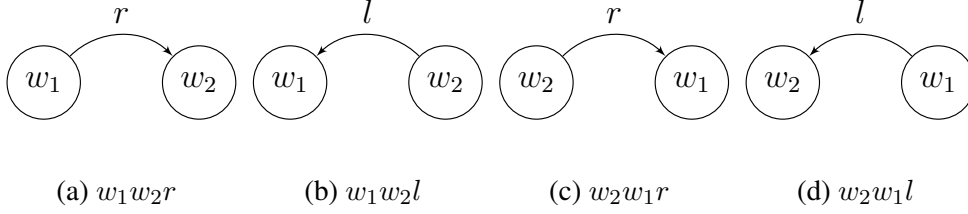


Figure 5.4: All the possible dependencies for two words. (a) and (b) describe Forward word order. In (a) w_1 precedes w_2 , and the dependency is right pointing r , i.e, the right-most word w_2 is the dependent. Word order in (b) is identical to (a) but the dependency direction is left pointing l which makes w_1 the dependent and w_2 the head. In (c) w_2 precedes w_1 and the dependency is right pointing. (d) has the similar word order to (c) but the dependency is left pointing.

order for the head and dependent. In the dependency $d(w_1, w_2, dir)$, any word that fills w_1 slot precedes w_2 and dependency direction dir marks the dependency arrow head that can be either left pointing l or right pointing r . For example, a left dependency $dir = 'l'$ means w_2 is the head word.

So a restricted kind of multigraph is necessary here, when there are at most two edges from one vertex to another, with one of these edges representing the dependency for one ordering of the words and the other edge the reverse ordering. We now give the formal definition.

For a bag-of-words X we construct a weighted directed multigraph $G_x = (V_x, E_x)$ as follows.

1. **NODES** The vertex set V_x consists of the following nodes:

- Each Base Noun Phrase (base NP) is represented by a distinct node corresponding to the head of the NP. We do this to be compatible with previous work (Wan et al., 2009, Zhang and Clark, 2011, Zhang et al., 2012) where base NPs are considered atomic units in the input sentence.
- Each other element in X (i.e. single words) is represented by a node.
- There is an artificial node called ROOT.

2. **EDGES** The edge set E_x is defined as follows:

- The subgraph consisting of all non-root nodes $G_x^w = (V_x - \{\text{ROOT}\}, E_x^w)$ is a complete directed multigraph, such that for any nodes $v_i, v_j \in V_x - \{\text{ROOT}\}$, there are two weighted edges (v_i, v_j) and two weighted edges (v_j, v_i) ; in each pair one is labelled with an l (for forward word order where the dependent is on the left side of head) and the other with r (for reverse word order where dependent is on the right side of

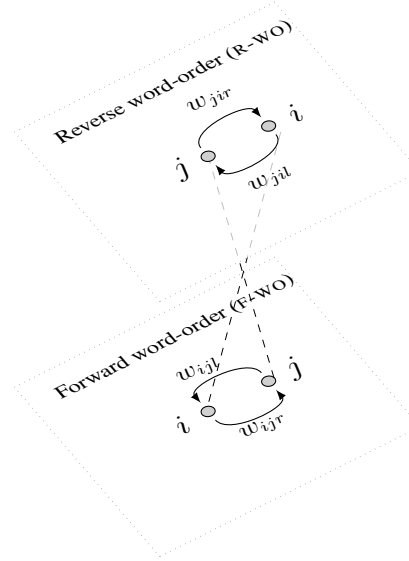


Figure 5.5: A complete weighted digraph for two words; it specifies word-order and dependency direction at the same time.

head). Figure 5.5 demonstrates such a subgraph consists of two nodes.

- From the artificial node ROOT , there are two outgoing edges to each other node $v_i \in V_x - \{\text{ROOT}\}$. Again, one of each pair is labelled with an l and the other with r .

Following Wan et al. (2009), the first structure we assign to the words is the straight MST, not yet including linguistic constraints. What we need then is an objective function and a set of constraints for us expressed via ILP. The spanning tree T of an undirected graph G is a subgraph that is a tree which includes all of the vertices of G . This definition asks for a connected graph with two limitations: (a) no cycle at all; and (b) all the vertices should be present. These limitations can be expressed by ILP constraints. If we want this tree to be an MST, we are looking for a set of edges so that the sum of the weights will be maximised (in the case of using cost instead of weight, we go for minimisation). In the parsing work, the ILP is a straightforward reformulation of CLE since any sentence can be mapped to a graph. However, when it comes to text generation, the representation of the bag-of-words becomes a multigraph as described before. This requires just a small adaptation to the ILP formulation of the CLE by imposing constraints that only permit one edge between any two vertices to get a tree. The following constraints are parallel to those of Riedel and Clarke (2006), and outlined in Section 5.2.2 adapted for our multigraph.

5.4 Defining an Objective Function

The objective function that obtains an MST from a graph G_x derived from a bag of words X as defined in Section 5.3 is defined as follows:

$$\max : \sum_{k \in \{l, r\}} \sum_{m=0}^N \sum_{n=0}^N s_{mnk} x_{mnk} \quad (5.5)$$

where m and n are indices that we arbitrarily assigned to words in the input, k is dependency direction (see EDGES in Section 5.3), and s_{mnk} is the score for edge d_{mnk} (We discuss estimating s_{mnk} in Chapter 6). There are $N + 1$ nodes in V_x with N words ($|X| = N$) and the artificial node ROOT that is indexed with 0. In the objective function (5.5) x_{mnk} is a binary variable that specifies whether the edge mnk participates in the final MST or not. ILP constraints below define how these variables relate to each other.

5.5 Enforcing Tree Structure

The optimisation result must be a spanning tree that contains all the words; a linguistic interpretation is a dependency tree. Therefore, both linguistic constraints and tree constraints must be present in the problem definition. Tree constraints ensure the solution is a valid MST. No self-loops, no cycles, and a unique parent head for individual child modifiers are instances of such constraints. Linguistic constraints guarantee that the given MST is a valid dependency tree, e.g., that the root of the tree is always ROOT, etc. The rest of this subsection discusses the construction of these constraints.

CONNECTIVITY All the nodes must be present in the final tree. The definition of connectivity for an N -node graph asks for $N - 1$ edges. This feature is simply represented by a single constraint.

$$\sum_{k \in \{l, r\}} \sum_{m=0}^N \sum_{n=0}^N x_{mnk} = N \quad (5.6)$$

Equation (5.6) is necessary to satisfy the connectivity, but is not sufficient. Combining this constraint with the NO-CYCLE constraint guarantees the connectivity.

NO-CYCLE There is no such single constraint that forbids cycles in a graph. For a graph of size N , all the possible cycles with length 2 to N must be identified.

Then, each individual cycle would be described by one constraint.

For a graph G_x with $|V_x| = N$, cycles of length 2 would be prevented by constraints of the form:

$$x_{ijl} + x_{ijr} + x_{jil} + x_{jir} \leq 1$$

Cycles of length 3 would be prevented by constraints of the form:

$$\begin{aligned} x_{13l} + x_{13r} + x_{31l} + x_{31r} &\leq 1 \\ x_{32l} + x_{32r} + x_{23l} + x_{23r} &\leq 1 \\ x_{12l} + x_{12r} + x_{21l} + x_{21r} &\leq 1 \end{aligned} \tag{5.7}$$

$$\begin{aligned} x_{12l} + x_{12r} + x_{13l} + x_{13r} + x_{23l} + x_{23r} \\ + x_{21l} + x_{21r} + x_{31l} + x_{31r} + x_{32l} + x_{32r} \end{aligned} = 2 \tag{5.8}$$

See Figure 5.6 for a sample graph of size 3. The first set of constraints avoids cycles of length 2 by picking each pair of nodes and enforces choosing at most one them at a time (inequalities in Equation (5.7)).

The number of cycle-forbidding constraints for a graph of size N is calculated in Equation (5.9).

$$CycleCount(N) = \sum_{i=2}^N \binom{N}{i} \tag{5.9}$$

The number of constraints of this type increases exponentially with the increase in the graph size.

The exponential number of constraints can lead to a dramatic increase in the processing time. This approach seems even less favourable considering that there are numerous solutions that are cycle-free by themselves, before applying any cycle-forbidding constraints. Therefore, we follow Riedel and Clarke (2006) by adding cycle constraints incrementally. In that case all the constraints except the cycle-forbidding ones are provided to the solver to find the best tree. If a cycle is found in a solution, then a cycle constraint is added accordingly until a tree is found. In Section 6.4, the iterative constraint addition is depicted in the ILP module in Figure 6.6.

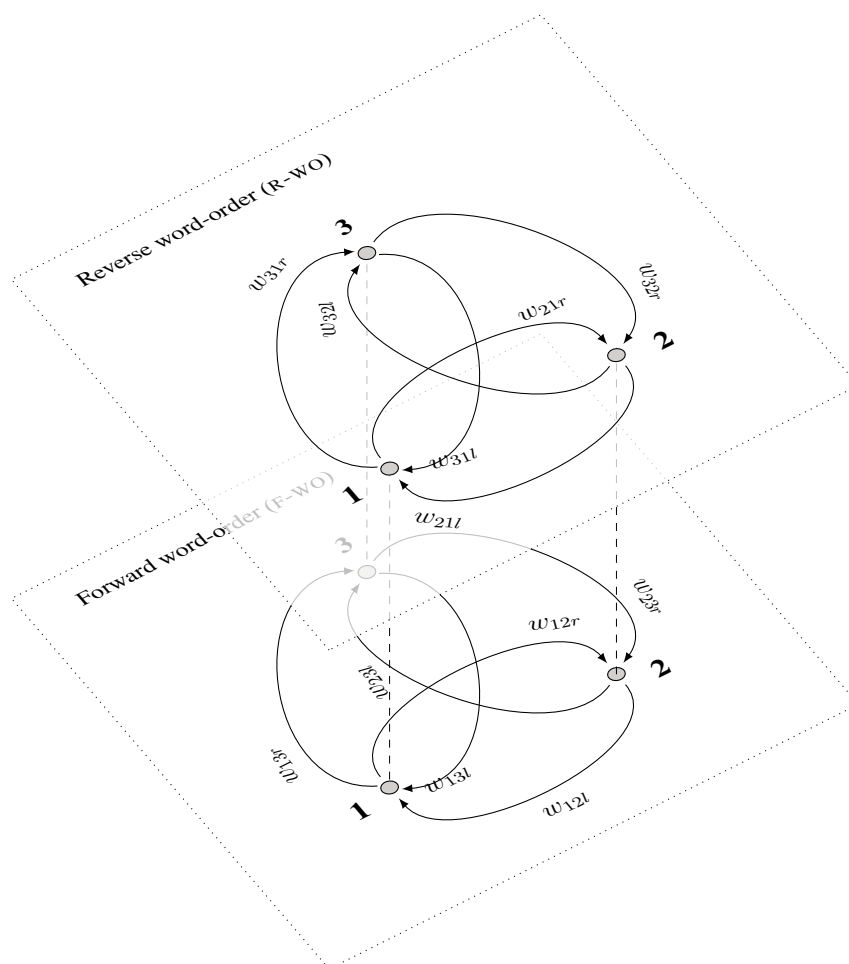


Figure 5.6: A complete weighted digraph for three words with word order and dependency direction specified at the same time

NO-SELF-LOOP-ON-THE-NODES Self-loops are cycles of length 1 that must be avoided as well. Equations of type (5.10) prevents self-loops in the graph.

$$\sum_{k \in \{l, r\}} \sum_{n=0}^N x_{nnk} = 0 \quad (5.10)$$

5.6 Imposing Linguistic Constraints

5.6.1 A Hard Encoding

Constraints introduced in this section transform a tree into a valid dependency tree with respect to structure and balance it using linguistically driven information. One of the structural properties of a dependency tree is the presence of a node called ROOT. This node has the same function as a root node in general trees, e.g. a traversal can be started from that node. However, it has some specific features that are described by constraints. Recall that this is assigned index 0 in the ILP representation.

Another structural property of a dependency tree is that the branching factor for each node depends on the word it represents. Linguistically speaking, some categories of words can be the modifier of some other, e.g., a preposition cannot be the modifier of a determiner and vice versa. Therefore, there must be some constraints that assign modifier(s) of an acceptable category to each given head word. Figure 2.15, which we used to illustrate Wan et al's motivation for including linguistic factors, shows the importance of imposing branching factor constraints on words in the process of assigning dependents to each headword.

5.6.1.1 Dependency Tree Constraints

ROOT is the head node for one and only one node (ROOT-HEAD) This restriction that sets the branching factor of ROOT to 1 is implemented using one constraint. Equation (5.11) expresses that amongst all the edges that leave the ROOT node, only one can be present in the final tree.

$$\sum_{n=1}^N x_{0nr} = 1 \quad (5.11)$$

$$(5.12)$$

Root shouldn't be a modifier (ROOT-NO-MODIFIER) Equation (5.13) disallows the edges with root node marked as their modifier from participating in the final tree.

$$\sum_{n=1}^N x_{n0r} + x_{0nl} = 0 \quad (5.13)$$

Unique head for individual modifier (ONLY-ONE-HEAD) Each individual node can be considered as modifier for only one head. Equation (5.14) represents N separate constraints, one for each possible value of n .

$$\forall_{n \in \{1,2,\dots,N\}} \sum_{m=1}^N x_{mnl} + \sum_{m=1}^N x_{nmr} = 1 \quad (5.14)$$

5.6.1.2 Branching Factor Constraints

As discussed in Section 2.3.2, each node's arity i.e. number of modifiers or branching factor as termed in graph theory, affects the shape of a tree and consequently the quality of the regenerated string. In this work, we introduce constraints to keep the branching factor for each node within a linguistically reasonable range. These constraints would be defined based on the head word's PoS tag. Arity for nouns and verbs are calculated based on a data-driven approach which allows a customised upper and lower bounds for arity per word. Coordination conjunctions, prepositions and punctuation get their distinct arity boundaries since they have different grammatical roles.

Nouns and verbs as head of a dependency (NOUN-VERB-ARITY) Since the produced spanning tree represents a dependency tree, we can include some linguistic information to avoid implausible dependency trees. For example consider a tree for an n word sentence where the i th takes all the other words as its dependents which (1) exceeds its own maximum branching factor and (2) violates other nodes' minimum number of dependents requirement (See Figure 5.7)⁸. In this section we impose constraints that are hard but not tight to prevent this: we enforce minimum and maximum numbers of dependents, depending on the part of speech of the word.

⁸In the previous constraint one modifier for ROOT is enforced. Therefore one of the words becomes the modifier of the root and the rest become the modifiers of this individual

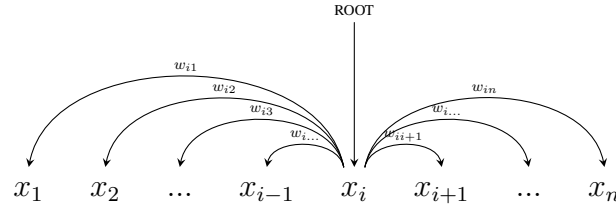


Figure 5.7: An example of a dependency tree where x_i has violated its maximum branching factor and caused some of the other nodes not to have the expected number of dependents.

Google syntactic n-grams (Goldberg and Orwant, 2013) is the resource we use to determine minimum and maximum acceptable branching factor for nouns and verbs. Google syntactic n-grams is a dataset of over 10 billion distinct counted dependency-tree fragments extracted from a corpus of 3.5 million English books, and so is a promising resource to estimate a branching factor. They defined a syntactic n-gram as a rooted connected dependency tree, which is a subtree of a dependency tree over an entire sentence. The n-gram also indicates the relative word order although neither the distance between the words, nor any missing material between the nodes is included. They produced these trees by training a PoS tagger and dependency parser on several corpora including WSJ Penn, Brown corpus and the Question Treebank. A reimplement of a beam-search shift-reduce dependency parser (Zhang and Clark, 2008) performed the parsing using the feature-set introduced in Zhang and Nivre (2011).

For the purpose of setting a minimum for right and left modifiers per head word, we looked at the unlexicalised version of dependencies. Since most of the entries have an instance of dependency with one modifier on one side and no modifier on the other side, the minimum number of modifiers tends to be 0 almost all the time which does not seem to be helpful in our case. Therefore, we picked the dependency entry with the greatest frequency as the most frequent template for valency and call it the MODE template. We use the number of left modifiers for such entries (MODEL) as the minimum number of modifiers on the side of the head word and similarly the number of the right modifiers for the most frequent dependency (MODER), to specify the minimum for the number of the right modifiers. Other useful extracted values are maximum number of left modifiers (MAXL) and maximum number of right modifiers (MAXR). These two values as implied by their names indicate the maximum number of modifiers on each side of the headword. The last two measures were not necessarily extracted from a single dependency entry e.g. the most frequent one. In Table 5.1, we present the entry for the verb *wake*:

Equation (5.15) uses these values to limit the branching factor of noun or verbs in bag-of-words X . Taking the set of head words with noun/verb PoS tags in the sentence as follows,

Head word	# mode template	MODEL	MODER	MAXL	MAXR
wake	179298	1	1	6	7

Table 5.1: Mode and maximum values of the branching factor for WAKE extracted from google syntactic n-gram.

$$H = \{index(head) : head \in X \cup head.pos = N \cup head.pos = V\}$$

We apply the limits over these heads.

$$\begin{aligned}
\sum_{h \in H} \sum_{m=1}^N x_{mhl} &\leq maxL \\
\sum_{h \in H} \sum_{m=1}^N x_{mhl} &\geq modeL \\
\sum_{h \in H} \sum_{m=1}^N x_{hmr} &\leq maxR \\
\sum_{h \in H} \sum_{m=1}^N x_{hmr} &\geq modeR
\end{aligned} \tag{5.15}$$

For example, if *wake* is the *i*th word in a sentence, its left and right valency constraint pairs are defined by inequalities in (5.16).

$$\begin{aligned}
\sum_{m=1}^N x_{mil} &\leq 6 \\
\sum_{m=1}^N x_{mil} &\geq 1 \\
\sum_{m=1}^N x_{imr} &\leq 7 \\
\sum_{m=1}^N x_{imr} &\geq 1
\end{aligned} \tag{5.16}$$

These enforce that *wake* must have at least one dependent on each side (subject on its left and PRP or PR on its right) and no more than 6 on its left and 7 on its right. In case of base-NPs, recall that they are treated as an atomic unit: the noun as a head word has already taken its immediate modifiers. Therefore, it is linguistically meaningless to dictate an extra branching factor for such a node and force it to choose some dependents as its modifiers. As result, we simply don't put any constraint for base-NPs.

Coordinating conjunctions as head of a dependency (COORD-VALENCY) Another category of words that we can define valency for is coordinators. Riedel and Clarke (2006) prepared a list of symmetric and asymmetric coordinators for German. These two lists are consulted every time a coordinator is available then suitable valencies are applied. (5.17) and (5.18) provide the branching factor for each conjunction token h that forms a symmetric and asymmetric coordination, respectively. Taking H as the set of symmetric coordinators, the constraints would be defined as:

$$H = \{index(head) : head \in X \cup head.pos = conj_{symmetric}\}$$

$$\sum_{h \in H} \sum_{m=1}^N x_{mhl} \geq 1 \tag{5.17}$$

$$\sum_{h \in H} \sum_{m=1}^N x_{hmr} = 1$$

Taking H as the set of asymmetric coordinators, the constraints would be defined as:

$$H = \{index(head) : head \in X \cup head.pos = conj_{asymmetric}\}$$

$$\begin{aligned}
\sum_{h \in H} \sum_{m=1}^N x_{mhl} &= 0 \\
\sum_{h \in H} \sum_{m=1}^N x_{hmr} &\geq 2
\end{aligned} \tag{5.18}$$

Branching factor for prepositions (PREP-VALENCY) Considering the way prepositions are used in English grammar, we bound prepositions to have only one dependent on their right side and none on their left.

$$H = \{index(head) : head \in X \cup head.pos = prep\}$$

$$\begin{aligned}
\sum_{h \in H} \sum_{m=1}^N x_{mhl} &= 0 \\
\sum_{h \in H} \sum_{m=1}^N x_{hmr} &= 1
\end{aligned} \tag{5.19}$$

Branching factor for punctuation (PUNC-VALENCY) Unlike the Stanford dependency parser (De Marneffe et al., 2006), we let punctuation participate in dependencies but only as modifiers. Equation (5.20) forbids punctuation to serve as heads while equation (5.21) ensures they are modifying one head word at a time.

$$H = \{index(head) : head \in X \cup head.pos = punct\}$$

$$\sum_{h \in H} \left(\sum_{m=1}^N x_{mhl} + \sum_{m=1}^N x_{hmr} \right) = 0 \tag{5.20}$$

$$\sum_{h \in H} \left(\sum_{m=1}^N x_{mhr} + \sum_{m=1}^N x_{hml} \right) = 1 \tag{5.21}$$

5.6.2 A Flexible Encoding

In the dependency tree creation task, each word's branching factor must take into account the others. The objective function along with all of the constraints defined above are aligned with this goal. As a side effect, there are some cases where NOUN-VERB-ARITY constraints cause the ILP solver to fail in locating a feasible solution. Consider the case where we have a sentence of length n and one of the nodes has a large minimum arity such as $\frac{n}{2}$. If the rest of the words have a moderate branching factor, e.g. 0 or 1, the dependency tree will be generated smoothly; however, if there exists at least one more word in the sentence that expects at least $\frac{n}{2}$ modifiers, then the ILP problem becomes an infeasible one. This problem occurs because the sum of the imposed minimum branching factors in that sentence exceeds $n - 1$. Another case would be the one where all the words have a small value as their minimum branching factor. If the upper bounds are also small, so that the sum of the upper bounds do not reach to $n - 1$, and CONNECTIVITY constraints cannot be satisfied. Consequently the problem would be an infeasible one.

One naive solution would be to allow 0 to be the minimum number of dependents and assign the largest observed valency as the upper bound. The program will always find a solution, but then the constraints are not word specific. Moreover, assigning a minimum and maximum number of modifiers to each word, as the accepted range for the number of modifiers, implicitly implies that there is no superiority amongst these different branching factors; presumably, all the valid solutions are equally good. However, that is not true.

To address this issue, we need to change our perspective to develop a novel way of describing this fuzzy boundary defined by each word's valency. This new approach would rank the feasible solutions by incorporating the difference between each node's branching factor and its reference valency. This reference valency is preferred to be word specific, in contrast to one valency for all words. It is also necessary to take into account all the words' valencies during the MST construction — valencies selected dynamically based on the available slots — and not just by looking at the appointed reference.

5.6.2.1 Multi-objective Optimisation

We first review some relevant literature on incorporation of constraints into an objective function before applying them to our problem. The problem of constraint satisfaction for combinatorial problems is well studied in mathematics (Bertsekas, 1982), operational research (Brailsford et al., 1999) and computer science, especially Evolutionary Computation (Smith and Coit, 1997, Michalewicz, 1995). Generally there are two ways of dealing with infeasible areas in optimisation. The first one is single objective optimisation. It introduces the infeasible areas by one

or more constraints. This is our approach in Section 5.6.1 to make an MST for the input graph. The other solution is redefining the constraint as an objective and then incorporating it into the main objective function as a penalty term. The first method requires the constraints to strictly distinguish the feasible and infeasible areas. These boundaries restrict the solver to search the feasible areas to find the optimum solution.

The second method is useful for the cases where the infeasible areas are hard to precisely describe by mathematical functions. Therefore, an estimate of the constraint is defined in the form of an objective function so that the estimation gets closer to the original value as it is maximised/minimised. This new term is added into the main objective function as a penalty term. These two terms can sometimes move in opposite directions, therefore there is not necessarily one global optimum solution for the problem but a set of Pareto-optimal solutions. The penalty term discourages the solver from exploring the infeasible areas by applying a large penalty which subsequently increases the value of the objective function. In case of minimisation, all the infeasible solutions' score are larger than the feasible ones. To adjust the penalty term, a new parameter λ is introduced to scale the penalty term. This scaling defines the severity of the penalty. The main challenge here is to define the penalty term so that it approximates the constraints as well as possible. The other challenge is selecting the value of λ , because the optimal solution frequently lies on the boundary of the feasible region (Smith and Coit, 1997).

Multi-objective optimisation problems are conventionally solved using the Lagrange Multiplier method. Assume we want to solve the optimisation problem below.

$$\begin{aligned} \text{minimize:} \quad & F(x, y) = x^2y - \ln(x) \\ \text{subject to:} \quad & 8x + 3y = 0 \end{aligned}$$

First we need to write y as a function of x that is, $y = -\frac{8}{3}x$ and then create a single objective function $f(x)$.

$$f(x) = F(x, -\frac{8}{3}x) = -\frac{8}{3}x^3 - \ln(x)$$

We then take the derivatives to

$$\begin{aligned} f'(x) &= -8x^2 - \frac{1}{x} \\ f''(x) &= 16x + \frac{1}{x^2} \end{aligned}$$

$f(x) = -\frac{1}{2} > 0$ so $x = -\frac{1}{2}$ is the relative minimum of f and therefore, $y = \frac{4}{3}$.

The Lagrange multiplier method is defined so that, if $F(x, y)$ is the objective function and $g(x, y)$ is a constraint, then $H(x, y)$ will be $F(x, y) + zg(x, y)$. If (a, b) is the relative extremum of F subject to $g(\cdot) = 0$, then there exists some value $z = \lambda$ such that

$$\frac{\partial H}{\partial x}|_{a,b,\lambda} = \frac{\partial H}{\partial y}|_{a,b,\lambda} = \frac{\partial H}{\partial z}|_{a,b,\lambda} = 0 \quad (5.22)$$

In the above equation, λ is also known as the Lagrange multiplier.

On the other hand, Evolutionary methods such as Genetic Algorithms (GA) and Particle swarm optimization (PSO) tend to solve optimisation problems numerically. Contrary to traditional models such as simulated annealing, these algorithms have a set of random starting points. For example in GA, each starting point is a proposed solution and is called a genome or chromosome. A set of random genomes forms the first generation. In each iteration, genomes of the current generation are ranked by their fitness score. The selection algorithm picks some genomes for populating the next generation and discards the rest.

The terminology used in evolutionary processing is slightly different from the Lagrange Multiplier method we just discussed. $F(\cdot)$ and $g(\cdot)$ are called the fitness function and penalty function respectively. λ is known as the severity of the penalty. Both penalty function and severity multiplier are problem-dependent and need to be tuned. Generally, penalty functions are categorised into three main classes: static, dynamic, and adaptive (Smith and Coit, 1997). Static penalty functions penalise the objective function by applying a constant penalty to the infeasible solutions. Dynamic penalty functions, on the other hand change the severity of the penalty by time/generation. In other words, it is likely for the program to search the infeasible areas at the beginning and it becomes less likely as evolution occurs. Adaptive penalty functions are meant to penalise the solution based on the success of the search within an interval. This way, the penalty function can guide the search by looking at multiple generations and the search trajectory.

Graph problems such as the MST problem are favourite optimisation problems.

They have been modelled as both single-objective and multi-objective problems in the evolutionary computing field. We will briefly explain some models used for solving the Minimum Vertex Cover (MVC) and the MST problem. To start with, we look at the research carried out by Neumann and Wegener (2007). They argued that with evolutionary algorithms, using a multi-objective variant of a single-objective problem can lead to a more efficient optimisation process. They tested their claim on the MST problem for weighted undirected graphs. Their test set was composed of a set of randomly chosen dense graphs with reasonable size. They defined the fitness function for the multi-objective approach as $f(s) = (c(s), w(s))$, where s is a subgraph proposed as the solution, $c(s)$ indicates the number of connected components in s , and $w(s)$ the subgraph's total weight.

Khuri and Bäck (1994) applied GA as another approach to find the MVC for a given graph $G = (V, E)$. The MVC is characterised as a set of vertices V' where $V' \subseteq V$ such that $\forall (i, j) \in E, i \in V'$ or/and $j \in V'$. In this work, edges are represented using a binary adjacency matrix, such that e_{ij} is 1 if an edge exists between nodes i and j and 0 otherwise.

They encode the MVC solution as a binary string of length n where each bit represents a node. Therefore, each genome is a string with length n and the bits associated with nodes in the proposed cover are set to 1. A two-point cross-over is used to populate the new generation. The fitness function is defined as below:

$$f(x) = \sum_{i=1}^n (x_i + n(1 - x_i) \cdot \sum_{j=1}^n (1 - x_j) e_{ij})$$

The first term of $f(x)$ determines size of the potential cover V' and the second term of $f(x)$ penalises those V' s that are not covers.

In many graph theory problems, the solution is a specific sub-tree within a given graph. If the genomes encode the solution, there is high risk of missing good genomes during the search process: Because the GA operators are not designed for creating trees, the offsprings trees may not be related to their parents. To address this issue, weighted codings were introduced (Palmer and Kershenbaum, 1994) and can be applied to a number of combinatorial problems such as the traveling salesman problem (Julstrom, 1998) and the multiple container packing problem (Raidl, 1999) amongst others. Weighted codings define genomes to be strings of weights that temporarily bias parameters of the problem instance. A decoding algorithm identifies the structure a chromosome represents using the biased parameters, and the chromosome's fitness is that structure's fitness calculated with the original parameters. Searching the space of weights provides a search of the target problem's search space.

Palmer and Kershenbaum (1994) used weighted-encoding for the MST problem. It assigns each vertex a random weight from a uniform distribution. For a given node i , its weight w_i is included in the cost of all the participating edges. Therefore, the cost value for the edge connects vertex i to j would be:

$$c'_{i,j} = c_{i,j} + w_i + w_j \quad (5.23)$$

These biased costs identify the MST using Prim's algorithm (Prim, 1957) (see Cormen et al. (2001)). The original costs, however, are used for the genomes' fitness calculation.

For example, if (C) is the cost matrix for a 3 node graph, and W is the random weights array, then (C') is calculated using equation 5.23.

$$C = \begin{pmatrix} 0 & 17 & 72 \\ 17 & 0 & 42 \\ 72 & 42 & 0 \end{pmatrix}$$

$$W = \langle 16, 54, 35 \rangle$$

$$C' = \begin{pmatrix} 0 & 87 & 123 \\ 87 & 0 & 131 \\ 123 & 131 & 0 \end{pmatrix}$$

Raidl and Julstrom (2000) proposed using multiplication in (5.23) instead of addition. They applied it to Degree-constrained MST. a variation of MST where none of the nodes could have a degree exceeding k ($k \geq 2$).

Therefore, the edge costs are calculated as

$$c'_{i,j} = c_{i,j} \cdot w_i \cdot w_j \quad (5.24)$$

This way the decoding algorithm enforces constraints, so all the genomes represent feasible solutions. In other words, there is no need to discard, repair, or penalise invalid chromosomes.

As discussed above one standard way of dealing with constraints that are hard to describe and may cause many solutions to be infeasible is to reformulate them so that they could be added to the original objective function with a multiplier. This redefinition of the constraints as penalty terms has the benefit of directing the search to the optimal solution by keeping the distance from infeasible areas. The reformulation of a single objective to multi-objective seems a promising approach

to the NOUN-VERB-ARITY constraints. This means each node's branching factor would be as close as possible to its reference valency without imposing a hard constraint on each node that can cause potential failure.

Now we are going to use the idea of embedding the constraints into the objective function. We study the effect of this transformation by keeping the same optimisation method and other constraints unchanged.

5.6.2.2 Reformulation of MST to a Multi-objective Problem

To convert the single objective optimisation solution proposed in Section 5.3 to a multi objective optimisation, the first step would be redefining the NOUN-VERB-ARITY hard constraint as another objective function.

For a given word, let the reference valency be the weighted mean of all its observed branching factors in Google Syntactic n-gram. One way to encourage reasonable valencies in the MST is by minimising the distance between the ILP-assigned valency $v(n)$ and the reference valency $v_r(n)$.

$$\min : \sum_{n=0}^N \text{dist}(v(n), v_r(n)) \quad (5.25)$$

where N is the size of $Y \subseteq X$ such that Y contains the head words of the base NPs and the verbs of bag-of-words X . Converting NOUN-VERB-ARITY to the penalty term described in Equation (5.26) transforms our single-objective optimisation problem into a multi-objective problem. In this redefinition, we attempt to minimise two terms at the same time: the total weight of the tree and the sum of distances between the $v(n)$ and $v_r(n)$ terms. We define the *dist* function for a word n as $\frac{(v(n)-v_r(n))^2}{s}$, with v_r being the weighted mean \bar{v} , and s the weighted standard deviation that is used as the normalisation term for the differences. Therefore, the penalty function would be:

$$\lambda \sum_{n=0}^N \frac{(v(n) - \bar{v}(n))^2}{s} \quad (5.26)$$

As can be seen, the distance function is a quadratic function. This makes the program a quadratic optimisation problem.

As an example, consider sentence (5.4). The words that can serve as a head word are underlined. The word *book* is not marked as a head word, since it is the head of a Base Noun Phrases (base NPs) and base NPs are treated as atomic parts of the

left v	right v	# occurrence	right v	# occurrence	probability
0	0	61515	0	62533	78.53
0	1	13925	1	14694	18.45
0	2	1933	2	1987	2.49
0	3	308	3	349	0.44
0	4	41	4	41	0.05
0	5	12	5	12	0.01
0	9	11	9	11	0.01
1	0	266			
1	1	482			
1	2	12			
2	0	752	left v	# occurrence	probability
2	1	272	0	77745	97.63
2	2	42	1	760	0.95
2	3	41	2	1107	1.39
3	1	15	3	15	0.02

(a) (b)

Table 5.2: (a) Valency templates for the word *eat*. (b) Probability of right and left valencies are calculated.

	eat		cats		mice	
	left	right	left	right	left	right
\bar{v}	0.03	0.25	0.21	0.50	0.22	0.58
s	0.06	0.28	0.32	0.44	0.28	0.52

Table 5.3: Weighted mean (\bar{v}) and weighted standard deviation (s) of the left and the right valencies of the head words in sentence (5.4)

sentence based on the problem description. The reference valency for these head words must be calculated, and included in the penalty term.

(5.4) mice eat cats in this fiction book.

Table 5.2 (a) lists the right and left valencies for the word *eat*. These values are extracted by looking into each template (as in the example of Table 5.1) and counting the right and left modifiers. Tables 5.2 (b) contains right and left valency counts independently from one another. These numbers are used for calculating the marginal probability of each word, and serve as weights for the weighted mean and weighted standard deviation calculation. These numbers are listed in Table 5.3.

As an example, label the words *eat*, *cats*, *mice*, *book* with indices 1, 2, 3, 4, which then constitute our set of nodes Y , and let $m, n \in Y$ and $k \in \{l, r\}$. Following

Section 5.3, w_{mnk} is the weight of the edge between node m and n ; while y_{mnk} is the binary value that indicates the presence of the edge that connects words m to n . The penalty term that looks after valencies for the word *eat* is calculated, based on the summation in Equation (5.26), and the values for \bar{v} and s from the lefthand entries of Table 5.3, is:

$$\frac{(y_{12r} + y_{13r} + y_{14r} - 0.25)^2}{0.28} + \frac{(y_{12l} + y_{13l} + y_{14l} - 0.03)^2}{0.06} \quad (5.27)$$

The penalty term for the other two head words are calculated similarly. The result penalty term that will be added to the objective function as follows

$$\min : \sum_{k \in \{l, r\}} \sum_{m=0}^4 \sum_{n=0}^4 w_{mnk} y_{mnk} + \lambda \sum_{m=1}^3 \sum_{k \in \{l, r\}} \sum_{n=1}^4 \left(\frac{y_{mnk} - \bar{v}_{mk}}{s_{mk}} \right)^2 \quad (5.28)$$

where, \bar{v} and s represent weighted mean and standard deviation.

As can be seen the objective function in (5.28) is a quadratic objective function: x_{mnk}^2 . This reformulation makes the optimisation problem to be Quadratic Programming (QP).

So far we assumed the w_{mnk} will be calculated using formula (2.5). However, we can train a model to learn the weights. The details of the learning process for dependency weights through a machine learner will be discussed in the next chapter, followed by the experiment results of each of these frameworks and where they sit compared to one another and with respect to baselines that do not include linguistic information.

5.7 Conclusion

The focus of this chapter was on string regeneration from a bag-of-words. Our high level goal is to find the most probable dependency tree for a bag-of-words and linearise it to get a sentence. Along the lines of Wan et al., we mapped the bag-of-words to a multi-graph with edge weights being the probability of a dependency relation between any two given nodes. Then we framed the problem of finding a dependency tree to finding an MST for such a multi-graph with the ILP framework, to allow constraints on tree structure to be included declaratively. To improve the quality of the dependency tree, we introduced linguistic constraints. In our first formulation, these are defined as hard constraints, and may lead to infeasible optimisation problem. To allow more flexibility for such linguistic

constraints, we extend the single objective optimiser ILP framework to a multi objective optimisation QP by redefining such constraints as additional objectives.

In Chapter 6, we carry out experiments on the sentence regeneration problem using these frameworks. We propose a method for learning the edge weights on the graph, analogous to the MIRA approach for dependency parsing used by McDonald et al., and then compare the ILP and QP frameworks, as methods for incorporating linguistic constraints, against various baselines.

Chapter 6

Applying the ILP Framework

6.1 Introduction

Chapter 5 described the Integer Linear Programming (ILP) framework for string regeneration that assigns a Minimum Spanning Tree (MST) to a bag-of-words as an intermediate structure prior to the linearisation. This framework relies on an objective function and constraints to guide the optimiser, either ILP or QP. ILP uses the hard-coded constraints to assign a MST, whilst QP intends to improve ILP by embedding the hard-coded NOUN-VERB-ARITY constraints into the objective function in the form of penalty terms i.e. soft constraints. The framework of the previous chapter just provides a method for finding an MST. Another necessary component is a way of assigning weights to the edges of the graph over which the MST is calculated. One approach — the one taken by Wan et al. — is to just use the Maximum Likelihood Estimate (MLE) calculated over a large dependency treebank (see Equation 2.5). However, these may not be the most useful weights for finding a good MST: rather, ones learnt specifically for the objective function.

The proposed ILP framework deployed hard-coded dependency weights which were calculated using MLE method over a large dependency treebank. In this chapter, we therefore review the Margin Infused Relaxed Algorithm (MIRA) (Crammer et al., 2006) which has been used for learning dependency weights in dependency parsing by McDonald et al. (2005a). We discuss how the learning process can be modified so that the model learns the precedence of dependency elements in addition to dependency weights, given a bag-of-words. We expect models using MIRA weights to rely less on gold standard dependency data than MLE-based weights do and as a result to be extendable to any other language that comes with a small dependency treebank, even a de-lexicalised one. We believe that such flexibility with the size of manually annotated data makes this method suitable for low-density languages.

Given the framework, then, for constructing a weighted graph from a bag-of-words and a method for finding an MST that can incorporate linguistics criteria, we carry out an empirical investigation into the following questions:

- RQ6.1** How successful is the ILP model in string regeneration compared to our baselines CLE and AB-based baselines, starting first with the MLE weights used by Wan et al.?
- RQ6.2** Does the presence of linguistically-motivated constraints contribute to the quality of the regenerated string?
- RQ6.3** Are soft constraints as implemented in our Quadratic Programming (QP) framework better than hard constraints as implemented in ILP?
- RQ6.4** Is MIRA capable of learning the precedence of the dependency element, in addition to the dependency weight?
- RQ6.5** Finally, can QP with MIRA weights outperform our other proposed models that use MLE weights in terms of the quality of the re-generated text?

6.2 Related Work: Online Large Margin Learning

To briefly recap the work of Wan et al. (described in Section 2.3.2) that we use as our starting point, they represent the bag-of-words by a weighted complete di-graph; each dependency's weight was calculated as the log probability of that dependency in the Penn treebank using formula (2.5). These weights would then be used in guiding the Chu-Liu-Edmonds (CLE) algorithm for finding an MST as the intermediate representation. This representation would further assist with the linearisation. In chapter 5, we replaced the CLE algorithm with the ILP framework. In this chapter, we take a different approach to learning the dependency weights, rather than Wan et al.'s pure MLE estimates, mainly for the two following reasons: (1) it would be also expected to perform better to have weights tailored to the specific task, and (2) to pursue the initial motivation of this thesis, of improving text generation for minority languages.

Much recent research in parsing and string regeneration has opted for learning weights in a model using machine learning methods. MIRA is one of the learning algorithms that was reported to be effective on learning NLP-related features. This algorithm performs multi-class classification and learns model parameters by iterating through the training set items and classifying each item individually. For example, Bohnet (2009) used MIRA for syntactic and semantic parsing; this system, taking part in the CoNLL shared task in 2009, achieved the highest syntactic parsing for English and German.

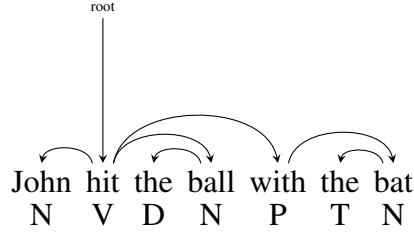


Figure 6.1: Dependency tree for sentence *John hit the ball with the bat..*

In this section, we describe MIRA as it is applied to learning dependency weights for parsing. This application can be modified relatively straightforwardly to suit our intended task. One of the first pieces of research that combined graph-based search for dependency parsing and learning dependency weights with machine learning methods is the work by McDonald et al. (2005b). In this section, we will illustrate MIRA with reference to this work.

Algorithm 2 gives an overview of an Online Large Margin Learner (McDonald et al., 2005a). The core of this algorithm is its updating strategy (line 4). The number of times that the learner iterates through the training dataset is indicated by N . T is the training set size. x_t and y_t are the gold standard dependency trees and the candidate dependency tree from training data, respectively. $w^{(i)}$ represents the features' weights at the i th update.

Algorithm 2 Generic online learning algorithm (McDonald et al., 2005a)

```

1   $i \leftarrow 0, w^{(0)} \leftarrow 0$ 
2  for  $n:=1$  to  $N$  do
3    for  $t:=1$  to  $T$  do
4       $w^{(i+1)} \leftarrow \text{update } w^{(i)} \text{ according to } \{x_t, y_t\}$ 
5       $i \leftarrow i + 1$ 
6    end for
7  end for
8  return  $\frac{\sum_{j=1}^{N \times |T|} w^{(j)}}{N \times |T|}$ 
```

The general update strategy is described as an optimisation problem. The components of this optimisation problem, e.g. objective function and loss function, are problem-specific. McDonald et al. trained a model using a supervised learner, MIRA, to learn dependency weights from a set of dependency features. The CLE algorithm (previously discussed in Algorithm 1 on page 42) was used as the search algorithm to find a spanning dependency tree.

Dependency parsing can be regarded as a classification problem in which each dependency tree is a class and the classifier's task is to find the appropriate tree for a given sentence x .

feature template	example
p-wrd, p-pos	p-wrd='hit', p-pos:'V'
p-wrd	p-wrd:'hit'
p-pos	p-pos:'V'
c-wrd, c-pos	c-wrd:'with', c-pos:'P'
c-wrd	c-wrd:'with'
c-pos	c-pos:'P'

(a) Basic Uni-gram Features

feature template	example
p-wrd, p-pos, c-wrd, c-pos	p-wrd:'hit', p-pos:'V', c-wrd:'with', c-pos:'P'
p-pos, c-wrd, c-pos	p-pos:'V', c-wrd:'with', c-pos:'P'
p-wrd, c-wrd, c-pos	p-wrd:'hit', c-wrd:'with', c-pos:'P'
p-wrd, p-pos, c-pos	p-wrd:'hit', p-pos:'V', c-pos:'P'
p-wrd, p-pos, c-wrd	p-wrd:'hit', p-pos:'V', c-wrd:'with'
p-wrd, c-wrd	p-wrd:'hit', c-wrd:'with'
p-pos, c-pos	p-pos:'V', c-pos:'P'

(b) Basic Big-ram Features

feature template	example
p-pos, b-pos, c-pos	p-pos:'V', b-pos:'D', c-pos:'P'
p-pos, p-pos+1, c-pos-1, c-pos	p-pos:'V', p-pos+1:'D', c-pos-1:'N', c-pos:'P'
p-pos-1, p-pos, c-pos-1, c-pos	p-pos-1:'N', p-pos:'V', c-pos-1:'N', c-pos:'P'
p-pos, p-pos+1, c-pos, c-pos+1	p-pos:'V', p-pos+1:'D', c-pos:'P', c-pos+1:'D'
p-pos-1, p-pos, c-pos, c-pos+1	p-pos-1:'N', p-pos:'V', c-pos:'P', c-pos+1:'D'

(c) In Between PoS Features

feature template	example
p-wrd, p-pos, c-wrd, c-pos, dir, dist	p-wrd:'hit', p-pos:'V', c-wrd:'with', c-pos:'P', dir:'R', dist:'2'
p-pos, c-wrd, c-pos, dir, dist	p-pos:'V', c-wrd:'with', c-pos:'P', dir:'R', dist:'2'
p-wrd, c-wrd, c-pos, dir, dist	p-wrd:'hit', c-wrd:'with', c-pos:'P', dir:'R', dist:'2'
p-wrd, p-pos, c-pos, dir, dist	p-wrd:'hit', p-pos:'V', c-pos:'P', dir:'R', dist:'2'
p-wrd, p-pos, c-wrd, dir, dist	p-wrd:'hit', p-pos:'V', c-wrd:'with', dir:'R', dist:'2'
p-wrd, c-wrd, dir, dist	p-wrd:'hit', c-wrd:'with', dir:'R', dist:'2'
p-pos, c-pos, dir, dist	p-pos:'V', c-pos:'P', dir:'R', dist:'2'

(d) Extended Features

Table 6.1: Features used in dependency parser (McDonald, 2006) replaced by p and c prefixes indicate parent node and child node respectively. wrd is the actual word and pos is the PoS tag of the given node. c-pos+1 represents the PoS tag to the right of the child node while c-pos-1 refers to the PoS tag to the left of the child node. The example column is illustrating such features based on the dependency tree in Figure 6.1.

If y is the dependency tree for x , it represents all the directed dependencies from word x_i to word x_j : $(i, j) \in y$. The dependency tree $s(x, y)$ score is calculated by accumulating the score of all the dependencies in y (Eisner, 1996):

$$s(x, y) = \sum_{(i,j) \in y} s(i, j) = \sum_{(i,j) \in y} \mathbf{w} \cdot \mathbf{f}(i, j) \quad (6.1)$$

where $\mathbf{f}(i, j)$ is a binary feature representation of (i, j) and \mathbf{w} is its corresponding weight vector.

The MIRA learning procedure is an optimisation problem as shown in definition (6.2). At each training epoch, the new weight vector $w^{(i+1)}$ must be the closest vector to current weight vector $w^{(i)}$ subject to satisfying the following constraint: using the new weights, the dependency tree score difference between the gold standard tree y and the incorrect tree y' must be less than or equal to the loss function $L(y, y')$.

$$\begin{aligned} \text{minimise:} \quad & \|w^{(i+1)} - w^{(i)}\| \\ \text{subject to:} \quad & s(x, y) - s(x, y') \leq L(y, y') \\ & \forall y' \in dt(x_t) \end{aligned} \quad (6.2)$$

where $dt(x_t)$ represents the set which contains all the possible dependency trees for the given sentence x . It can be seen from the third line in definition (6.2) that the number of constraints is highly dependent on the number of possible dependency trees $dt(x_t)$. Looking at all the possible dependency trees, $\forall y' \in dt(x_t)$ for each training item makes the learning process computationally expensive and quite slow. McDonald et al. reduced the number of constraints by only looking at the k -best trees. Their initial assumption was that the incorrect trees with the highest score are those ones that actually matters for the learner. The weight update using single best tree is as follows:

$$\begin{aligned} \text{minimise:} \quad & \|w^{(i+1)} - w^{(i)}\| \\ \text{subject to:} \quad & s(x, y) - s(x, y') \leq L(y, y') \\ & \text{where } y' = \operatorname{argmax}_{y'} s(x_t, y') \end{aligned} \quad (6.3)$$

Evaluating parsers that used MIRA with definitions (6.2) and (6.3) update schemes

individually validated their assumption by showing a negligible degradation in performance of k -best MIRA. Single-best MIRA refers to the case that the first best tree is only used for training purposes.

To clarify how the weights are updated using single-best MIRA, we give an artificial example. Figure 6.2 illustrates the gold standard tree y for a sample sentence x . Figure 6.3 shows the 1-best y' for x which differs from the gold standard in only one edge — this will thus be the loss function value. Therefore, the new weight vector will be the one that has the closest values to $w^{(0)}$, $\min \|w^{(i+1)} - w^{(i)}\|$, satisfying the following constraint:

$$\begin{aligned}
 s(x, y) - s(x, y') &\leq L(y, y') \\
 \sum_{(i,j) \in y} s(i, j) - \sum_{(i,j) \in y'} s(i, j) &\leq 1 \\
 \sum_{(i,j) \in y} \mathbf{w}^{(1)} \mathbf{f}(i, j) - \sum_{(i,j) \in y'} \mathbf{w}^{(1)} \mathbf{f}(i, j) &\leq 1 \\
 \sum_{(7,6) \in y} \mathbf{w}^{(1)} \mathbf{f}(7, 6) - \sum_{(5,6) \in y'} \mathbf{w}^{(1)} \mathbf{f}(5, 6) &\leq 1
 \end{aligned}$$

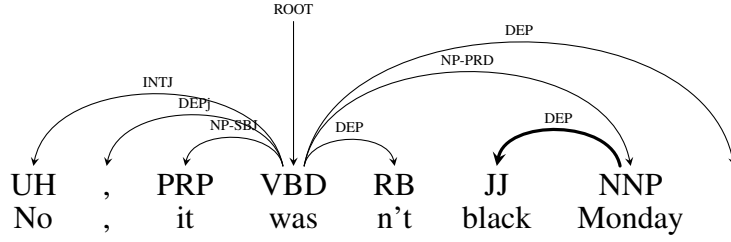


Figure 6.2: Gold standard dependency tree.

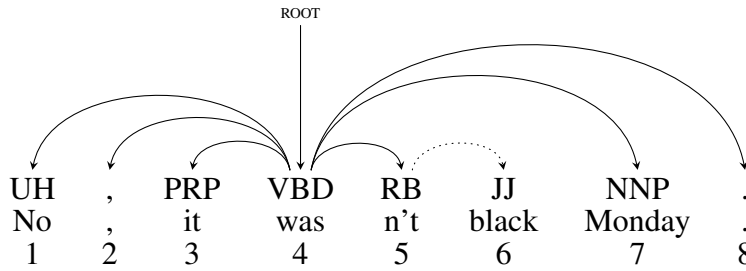


Figure 6.3: First best predicted parse tree.

The real world case of course has more incorrect dependencies like the tree depicted in Figure 6.4. As can be seen in Equation (6.4), single-best MIRA creates a more

complicated constraint to update the weights.

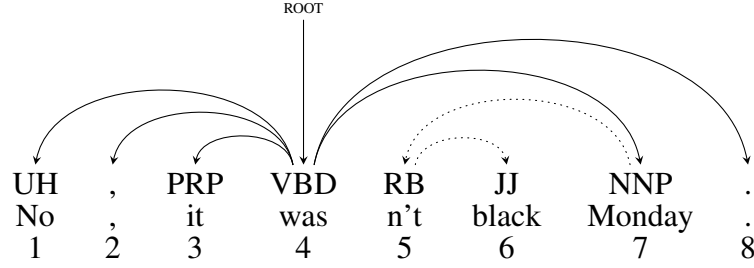


Figure 6.4: Second best predicted parse tree.

$$\begin{aligned}
 s(x, y) - s(x, y') &\leq L(y, y') \\
 \sum_{(i,j) \in y} s(i, j) - \sum_{(i,j) \in y'} s(i, j) &\leq 2 \\
 \sum_{(i,j) \in y} \mathbf{w}^{(1)} \mathbf{f}(i, j) - \sum_{(i,j) \in y'} \mathbf{w}^{(1)} \mathbf{f}(i, j) &\leq 2
 \end{aligned} \tag{6.4}$$

where,

$$\begin{aligned}
 \sum_{(i,j) \in y} \mathbf{w}^{(1)} \mathbf{f}(i, j) &= \sum_{(7,6) \in y} \mathbf{w}^{(1)} \mathbf{f}(7, 6) + \sum_{(4,5) \in y} \mathbf{w}^{(1)} \mathbf{f}(4, 5) \\
 \sum_{(i,j) \in y'} \mathbf{w}^{(1)} \mathbf{f}(i, j) &= \sum_{(5,6) \in y'} \mathbf{w}^{(1)} \mathbf{f}(5, 6) + \sum_{(7,5) \in y'} \mathbf{w}^{(1)} \mathbf{f}(7, 5)
 \end{aligned}$$

6.3 Feature Selection and Feature Extraction

As explained in the previous section, Margin Infused Relaxed Algorithm (MIRA) has been proved to be useful in learning dependency parsing (McDonald et al., 2005b). Therefore, we take this algorithm and adapt it, so that in addition to its original application, it will be capable of partial word-ordering: assigning precedence to one of the dependency elements, i.e. head or dependent.

The adaptation has to regard the differences between dependency assignment to a sentence and to a bag-of-words in both feature definition and feature extraction levels. For feature definition, features listed in Table 6.1 must be revised to assign a dependency based on the proposed order and direction, excluding any other word-order inclusive information. For example, neighbourhood information used

in Table 6.1(c) is not extractable from a bag-of-words. Similarly, the *dist* feature in Table 6.1(d) must be excluded from all the templates. Table 6.2 lists features used in our MIRA approach.

Now that we have defined features that can capture word order, we have to decide how to assign word-order at feature extraction time. As discussed in Section 5.3, there exist 4 possible dependencies given a pair of words. This forms the search space previously described in Figure 5.5. This search space consists of two planes that differ in word order: F-WO plane represents dependencies where participating pairs follow the gold standard word order, and R-WO plane represents dependencies where the participating pair have reverse word order compared to the gold standard.

At the training phase with the presence of the gold standard dependency, we only extract features from dependencies in the (F-WO) plane. This way we reinforce the expected word order per pair. However, for testing, R-WO features are included in addition to F-WO features. We call this feature set FR-WO.

Since we trained our model over F-WO features, it is expected that dependencies with similar word order to gold standard dependency d_g have the largest feature sets (as the likelihood of having seen the dependency during the training phase is much higher) whilst the ones from the R-WO suffer from sparsity as the likelihood of having seen such dependencies in the training data is lower. For further clarification on how feature sets for four possible dependencies differs for a given pair of words, Figure 6.5 shows the encoded features for $\{A, B\}$ pair extracted using FR-WO scheme for testing a model trained over F-WO features.

Assume that we know from labelled data that d_2 is the d_g and just by looking at the encoded dependencies, we can see that d_3 and d_2 must have similar dependency components — parent, child and direction — but different word order as they share a number of features (highlighted with purple). Similarly, d_1 and d_4 must be similar in terms of dependency component. Features that are unique to each dependency, belong to the Extended (i.e. the ones given in Table 6.2.c), as they capture both word order and dependency components. d_3 and d_4 have the least number of unique features which can be an indication of no such pair — with the exact word order and dependency components — having been observed during the training. This makes these two pairs less likely to be present in the final dependency tree.

In the rest of this chapter we go through the details of the training and the testing process and how all the modules discussed so far will be combined to form QP-MIRA that is capable of learning dependency weights and direction.

feature template	example
p-word, p-pos	p-word='hit', p-pos='V'
p-word	p-word='hit'
p-pos	p-pos='V'
c-word, c-pos	c-word='with', c-pos='P'
c-word	c-word='with'
c-pos	c-pos='P'

(a) Basic Uni-gram Features

feature template	example
p-word, p-pos, c-word, c-pos	p-word='hit', p-pos='V', c-word='with', c-pos='P'
p-pos, c-word, c-pos	p-pos='V', c-word='with', c-pos='P'
p-word, c-word, c-pos	p-word='hit', c-word='with', c-pos='P'
p-word, p-pos, c-pos	p-word='hit', p-pos='V', c-pos='P'
p-word, p-pos, c-word	p-word='hit', p-pos='V', c-word='with'
p-word, c-word	p-word='hit', c-word='with'
p-pos, c-pos	p-pos='V', c-pos='P'

(b) Basic Bi-gram Features

feature template	example
p-word, p-pos, c-word, c-pos, dir	p-word='hit', p-pos='V', c-word='with', c-pos='P', dir='R'
p-pos, c-word, c-pos, dir	p-pos='V', c-word='with', c-pos='P', dir='R'
p-word, c-word, c-pos, dir	p-word='hit', c-word='with', c-pos='P', dir='R'
p-word, p-pos, c-pos, dir,	p-word='hit', p-pos='V', c-pos='P', dir='R'
p-word, p-pos, c-word, dir	p-word='hit', p-pos='V', c-word='with', dir='R'
p-word, c-word, dir	p-word='hit', c-word='with', dir='R'
p-pos, c-pos, dir	p-pos='V', c-pos='P', dir='R'

(c) Extended Features

Table 6.2: Features used in the QP-MIRA model. p- and c -prefixes indicate parent node and child node, respectively. word is the actual word and pos is the PoS tag of the given node. c-pos+1 represents the PoS tag to the right of the child node while c-pos-1 refers to the PoS tag to the left of the child node. The example column used the dependencies in Figure 6.1.

Dependency	Alias	Encoded Dependency Features									
<div><div>A</div><div><div></div><div></div><div></div></div><div>B</div></div>	d_1	128	86	96	99	308	312	314	927		
		929	930	931							
<div><div>A</div><div><div></div><div></div><div></div></div><div>B</div></div>	d_2	66	70	72	138	140	142	148	34		
		635	636	637	638	639	640				
<div><div>B</div><div><div></div><div></div><div></div></div><div>A</div></div>	d_3	18	42	66	70	72	138	140	142	148	
<div><div>B</div><div><div></div><div></div><div></div></div><div>A</div></div>	d_4	320	344	68	308	312	314	341	722	727	

Figure 6.5: Some of the encoded features for 4 possible dependencies given two words A and B. These features are defined in Table 6.2 and extracted from a model trained over the F-WO plane of training data. We colour-coded the features for easier reference. All the purple features — shared between d_2 and d_3 — like 66, must be of basic Uni-gram and/or basic Bi-gram features, as these two dependencies share the same dependency direction: A is the head and B is the dependent; similar argument is valid for blue features for d_1 and d_4 . The features highlighted with orange and gray must be of the Extended feature group, since they are unique to the dependency. We did not highlight unique dependencies for d_3 and d_4 for readability purposes. Represented with fewer unique features indicates that the last two dependencies have no exact match in the training set, but still partially matched with some instances, e.g. 18 represents **p-pos**, **c-pos**, **dir** feature (see Figure 6.3 for feature sets prior to encoding).

6.4 Training

The preliminary step of training with MIRA is creating a repository of F-WO features using the above mentioned templates. This process is explained formally in Algorithm 3.

Algorithm 3 creates the gold feature repository by extracting features from the dependency treebank. This function is called once and its result is saved for later use.

```

1  function ExtarctModelFeatures
2  for tree  $t$  in  $TreeBank$  do
3     $goldDeps \leftarrow ReadDeps(t)$ 
4     $FeatureID \leftarrow null$ 
5    for dependency  $d$  in  $goldDeps$  do
6       $FeatureBank.FeatureID.Add(encode(unigramFeatures(d)))$ 
7       $FeatureBank.FeatureID.Add(encode(bigramFeatures(d)))$ 
8       $FeatureBank.FeatureID.Add(encode(extendedFeatures(d)))$ 
9    end for
10 end for
11  $FeatureBank.Scr \leftarrow null$ 
12 for featureID  $f$  in  $FeatureID$  do
13    $FeatureBank.Scr[f] \leftarrow 0$ 
14 end for
15 return  $FeatureBank$ 
16 end function

```

The *unigramFeatures*, *bigramFeatures*, and *extendedFeatures* methods follow their respective templates described in Table 6.2. Once the features along with their initial scores are saved in the *FeatureBank* repository, they would be accessible for further use.

The rest of the training process would be applying the steps for every sentence in the training set:

1. Shuffle the i th sentence s_i from training set i.e., generating a bag-of-words.
2. Create its complete graph in the F-WO plane and calculate each dependency weight by applying the dependency score formula (Equation (6.1)) to its features' score.
3. Apply the MST-ILP or MST-QP algorithm to the graph to get the MST tree.
4. Update the feature weights using single best MIRA algorithm.

Figure 6.6 gives a visual description of the training phase and resources involved. The preliminary step is drawn with a different color to be distinguishable from the

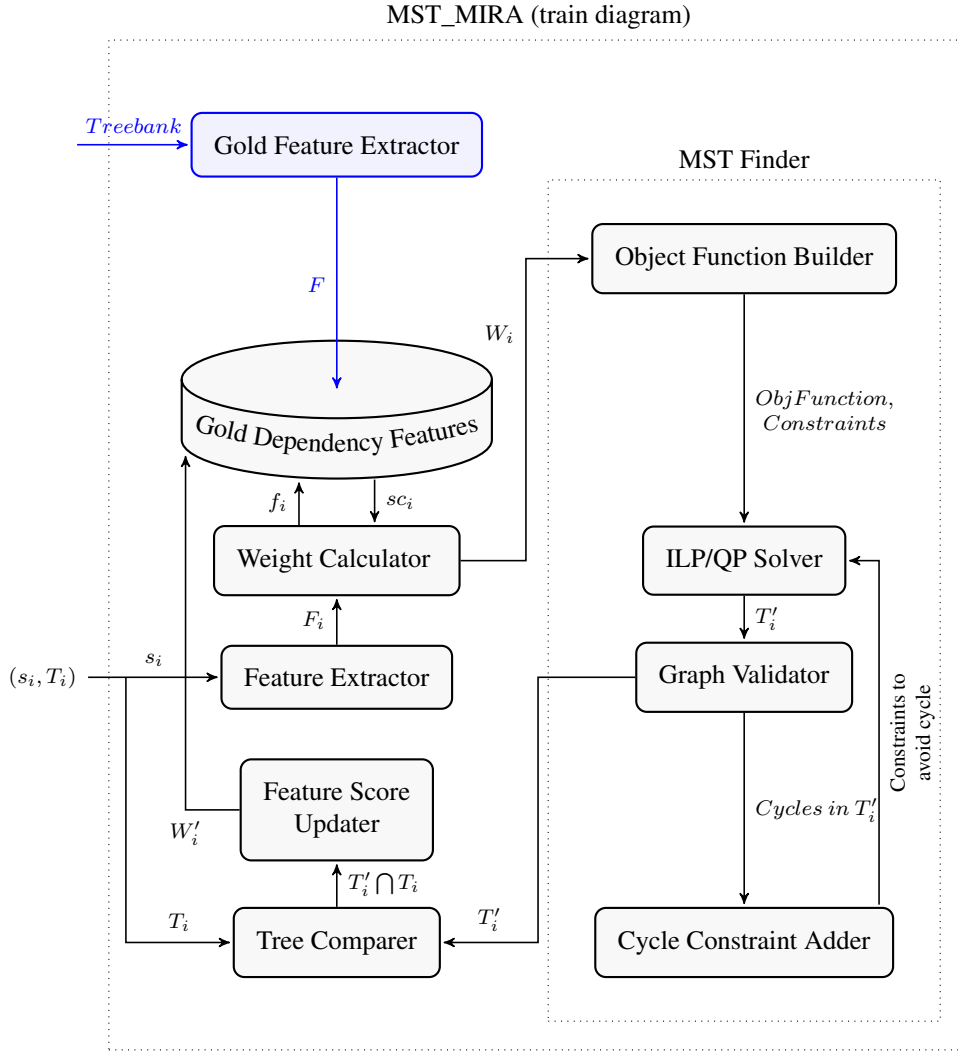


Figure 6.6: ILP-MIRA/QP-MIRA diagram

rest of the repeated training steps.

The first step is shuffling the words in the sentence and keeping the mapping for later reference. The feature extraction step then creates a complete weighted graph for each sentence and extracts features per dependency (F_i). For instance, the features extracted for all the four possible dependencies between the *root* and *eats* nodes in Figure 5.2 are shown in Table 6.3.

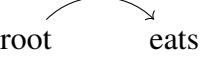
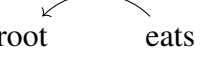


Dependency	Feature Group	Dependency Features
	basic uni-gram	pw:pp:root:rt, pw:rt, pp:rt, cw:cp:eat:vp, cw:eat, cp:vp
	basic bi-gram	pw:pp:cw:cp:root:rt:eat:vp, pp:cw:cp:root:eats:vp, pw:cw:cp:root:eats:vp pw:pp:cp:root:rt:vp, pw:pp:cw:root:rt:eat, pw:cw:root:eat, pp:cp:root:vp
	extended	pw:pp:cw:cp:d:root:vp:eat:vp:r, pp:cw:cp:d:rt:eats:vp:r, pw:cw:cp:d:root:eats:vp:r, pw:pp:cp:d:root:rt:vp:r, pw:pp:cw:d:root:rt:eat:r, pw:cw:d:root:eat:r, pp:cp:d:rt:vp:r
	basic uni-gram	pw:pp:eat:vp, pw:eat, pp:vp, cw:cp:root:rt, cw:root, cp:rt
	basic bi-gram	pw:pp:cw:cp:eat:vp:root:rt, pp:cw:cp:eat:root:rt, pw:cw:cp:eat:root:rt, pw:pp:cp:eat:vp:rt, pw:pp:cw:eat:vp:root, pw:cw:eat:root, pp:cp:vp:rt
	extended	pw:pp:cw:cp:d:eat:vp:root:rt:r, pp:cw:cp:eat:d:root:rt:r, pw:cw:cp:d:eat:root:rt:r, pw:pp:cp:d:eat:vp:rt:r, pw:pp:cw:d:eat:vp:root:r, pw:cw:d:eat:root:r, pp:cp:d:vp:rt:r
	basic uni-gram	pw:pp:eat:vp, pw:eat, pp:vp, cw:cp:root:rt, cw:root, cp:rt
	basic bi-gram	pw:pp:cw:cp:eat:vp:root:rt, pp:cw:cp:eat:root:rt, pw:cw:cp:eat:root:rt, pw:pp:cp:eat:vp:rt, pw:pp:cw:eat:vp:root, pw:cw:eat:root, pp:cp:vp:rt
	extended	pw:pp:cw:cp:d:eat:vp:root:rt:l, pp:cw:cp:eat:d:root:rt:l, pw:cw:cp:d:eat:root:rt:l, pw:pp:cp:d:eat:vp:rt:l, pw:pp:cw:d:eat:vp:root:l, pw:cw:d:eat:root:l, pp:cp:d:vp:rt:l
	basic uni-gram	pw:pp:root:rt, pw:root, pp:rt, cw:cp:eat:vp, cw:eat, cp:vp
	basic bi-gram	pw:pp:cw:cp:root:vp:eat:vp, pp:cw:cp:root:eats:vp,
	extended	pw:pp:cw:cp:d:root:rp:eat:vp:l, pp:cw:cp:d:root:eats:vp:l, pw:cw:cp:d:root:eats:vp:l, pw:pp:cp:d:root:rt:vp:l, pw:pp:cw:d:root:rt:eat:l, pw:cw:d:root:eat:l, pp:cp:d:root:vp:l

Table 6.3: Extracted features for the dependency (root,eats) from the complete graph presented in Figure 5.2 using FR-WO scheme.
Notation: pw: parent word, cw: child word, pp: parent PoS tag, cp: child PoS tag.

Algorithm 4 describes feature extraction, i.e. creation of weighted graph, for a set of sentences in more detail.

Algorithm 4 creates a feature set for a bag-of-words given that there is a dependency between any two given words.

```

1  function CreateCompleteGraph(goldSen)
2  for  $w1 = 0; w1 < \text{len}(\text{goldSen}); w1++$  do
3    for  $w2 = w1 + 1; w2 < \text{len}(\text{goldSen}); w2++$  do
4      for  $dir = 0; dir < 2; dir++$  do
5         $d \leftarrow \text{CreateDep}(w1, w2, dir)$ 
6         $ftrs \leftarrow \text{GenerateFeatures}(d)$ 
7         $F[w1][w2][dir] = \text{AssignFeatures}(ftrs)$ 
8        if (IncludeReverseFeatures) then
9           $d \leftarrow \text{CreateDep}(w2, w1, dir)$ 
10          $ftrs \leftarrow \text{GenerateFeatures}(d)$ 
11          $F[w2][w1][dir] = \text{FilterFeatures}(ftrs)$ 
12       end if
13     end for
14   end for
15 end for
16 return  $F$ 
17 end function

```

This algorithm uses the same notation for the dependency representation as in Section 5.3. That is, $F[w1][w2][d]$ means $w2$ follows $w1$ and dir decides the direction of the dependency, i.e. which word serves as head and which as dependent. The **for** loops in lines 2-4 are responsible for creating the complete graph. The first two loops pair up words ($w1, w2$) and the final loop assigns right and left dependencies to them. In lines 5-7, features for a given dependency d are extracted and then filtered to only contain those that are in the repository; see algorithm 5. This explains why there are different numbers of features for each dependency in Table 6.5. The default model consists of F-WO features unless *IncludeReverseFeatures* is set to *true* (line 8). R-WO features are made by creating a new dependency considering the new word order by swapping the order of $w1$ and $w2$.

Algorithms 4 and 5 appeared in Figure 6.6 under the modules named **Feature Extractor** and **Weight Calculator** respectively.

When the complete weighted graph is prepared for a given sentence, the graph is then passed to the optimiser module which is responsible for finding a subgraph that complies with the constraints discussed in Sections 5.5 and 5.6.1. This subgraph T'_i is then inspected for cycles. As long as T'_i does not represent a tree, the cycle-forbidding constraints are added incrementally; otherwise it would be sent back to the next step in the learning model. As previously discussed in Section 5.5, the

Algorithm 5 searches feature set and return features for a given dependency.

```

1 function FilterFeatures (ftrs)
2   ftrList  $\leftarrow$  null
3   for features f in ftrs do
4     if FeatureBank.FeatureID.contains(f) then
5       ftrList.Add([f, FeatureBank.Scr[f]])
6     end if
7   end for
8   return ftrList
9 end function

```

incremental constraint addition enables me to work with sentences of any length. However, the ILP solver response time drops due to the larger search space. To have a tractable training/testing time, we introduce two new parameters to our model. The first one is a timer that was added to the solver. This timer bounds the searching time for each given graph. If an optimal or a suboptimal solution can't be found within this period, the system would back off to the baseline. The other parameter limits the number of times **Cycle Constraint Adder** can be called. This parameter avoids the worst case scenario where the system creates all the cycle-forbidding constraints and adds them to the problem incrementally. Algorithm 6 explains the constraint addition process.

Algorithm 6 Incremental constraint addition

```

1: function GenerateILPDepTree (TreeConstraints, LinguisticConstraints,
   weights)
2: MAX  $\leftarrow$  maxIterations
3: timeOut  $\leftarrow$  N
4: ILP.Add(TreeConstraints)
5: ILP.Add(LinguisticConstraints)
6: ILP.AddObjFunction(weights)
7: res  $\leftarrow$  ILP.solve(timeOut)
8: if not(res == OPTIMAL or res == SUBOPTIMAL) then
9:   return null;
10: end if
11: while tree.HasCycle() and count < MAX do
12:   c  $\leftarrow$  tree.DetectCycle()
13:   ILP.Add(CycleConstraints(c))
14:   tree  $\leftarrow$  ILP.Solve()
15: end while
16: return tree
17: end function

```

Module **MST Finder** in 6.6 represents this process. **Tree Compare** finds the dissimilarity (as discussed in Section 6.2) between T' and T and uses it to update the feature weights in **Weight Calculator**. The weights are updated following the single-best MIRA rule (Equation (6.3)). The feature repository is updated with the new weights, prior to the processing of the next sentence. After the model reaches the point where the dependency scores become constant, the model is ready to be used/tested.

6.5 A Small-Scale Trial and Illustration

To re-generate from a bag-of-words, we need a feature repository that contains feature scores, an optimiser that is capable of making a tree whose nodes (the words) are partially ordered, and finally a lineariser. Figure 6.7 depicts such system. As expected most of the modules are shared between the training and testing systems.

As an illustration, we used a set of 50 random sentences to train and test MIRA with F-WO and FR-WO models, defined in Section 6.3, for comparing their results within the ILP hard constraint framework. Sentence (6.1) is the last sentence in this set. Its dependency trees for F-WO and R-WO (discussed in earlier in Section 6.3) are presented in Figures 6.8 and 6.9 respectively.

- (6.1) The White Sands missile range and CIA contractor Mitre Inc. were among the targets.

White Sands is contracted to *WS* for the sake of compactness. Appendix A provides a step-by-step picture of weight update per iteration. These dependency trees are the MSTs, given the weights in the tables, and only reflect the dependencies selected by the ILP module not the lineariser output. Tables 6.4 and 6.5 demonstrate the weight matrices for these two MSTs where the highlighted weights indicate the edges that are present in the corresponding trees.

6.6 Experiment Setup

6.6.1 General Details

In this section, we discuss three systems based on the frameworks defined in Chapter 5 and the weight-learning process defined in this chapter. The first two systems are based on the ILP and QP formulations described in Chapter 5, with the MLE edge weights used by (Wan et al., 2009). For the third system we take the

	root	range	and	Inc.	were	among	targets	.
root	0	1.475	-0.845	4.051	0.138	-0.384	0.205	0
range	3.580	0	2.719	7.666	3.122	3.116	3.534	3.580
and	1.436	3.141	0	5.894	1.333	1.053	1.514	1.436
Inc.	2.831	4.395	2.364	0	2.520	2.431	2.966	2.831
were	2.529	4.256	1.735	7.061	0	2.356	2.749	3.301
among	0.156	1.741	-0.689	4.317	-0.139	0	0.482	0.156
targets	2.961	4.890	2.162	7.139	2.680	2.505	0	2.961
.	0	1.475	-0.845	4.051	-0.311	-0.384	0.205	0
root	0	2.016	0.845	3.772	1.208	0.384	0.839	0
range	2.656	0	3.518	6.446	2.949	3.033	3.370	2.656
and	3.161	5.637	0	7.747	3.886	3.545	3.747	3.161
Inc.	2.490	4.684	4.361	0	2.800	2.929	3.250	2.490
were	2.539	4.760	3.486	6.760	0	3.282	3.410	4.085
among	0.620	2.857	1.465	4.900	0.964	0	1.703	0.620
targets	2.013	4.249	3.245	5.849	2.384	2.392	0	2.013
.	0	2.016	0.845	3.772	0.311	0.384	0.839	0

Table 6.4: Weight matrix for sentence (6.1) using F-WO model.

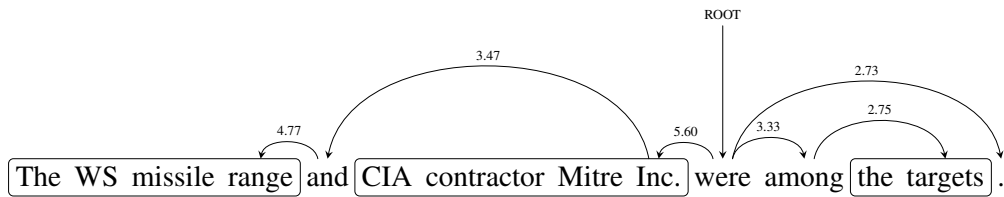


Figure 6.9: ILP Each Arc's label is the dependency weight associated with it, highlighted in Table 6.5.

	root	range	and	Inc.	were	among	targets	.
root	0	1.158	-1.053	3.555	0.144	-0.966	-0.268	0
range	3.462	0	2.379	7.038	2.977	2.355	3.282	3.462
and	1.118	2.444	0	5.03	0.795	0.153	0.466	1.118
Inc.	2.733	3.788	2.013	0	2.272	1.758	2.465	2.733
were	1.663	3.194	0.626	5.603	0	0.754	1.266	2.289
among	-0.728	0.495	-1.781	3.110	-1.220	0	-1.02	-0.728
targets	2.869	4.604	1.630	6.411	2.395	1.862	0	2.869
.	0	1.158	-1.053	3.555	-0.461	-0.966	-0.268	0
root	0	1.547	1.053	2.390	1.670	0.966	1.012	0
range	2.167	0	3.163	4.563	2.750	3.294	3.187	2.167
and	2.861	4.777	0	5.911	3.70	3.827	3.105	2.861
Inc.	1.746	3.314	3.473	0	2.207	2.693	2.758	1.746
were	2.135	3.921	3.221	4.943	0	3.332	3.204	3.385
among	1.425	3.464	2.478	4.428	1.878	0	2.750	1.425
targets	1.105	2.677	2.029	3.442	1.539	2.116	0	1.105
.	0	1.547	1.053	2.390	0.461	0.966	1.012	0

Table 6.5: Weight matrix for sentence (6.1) using R-WO model.

more flexible QP system and apply the MIRA weights to it; we refer to this system as QP-MIRA.

Since the plain ILP and QP systems borrow dependency weights from Wan et al. (2009)’s work, they require no training; the evaluation process for these systems involves applying the corresponding algorithms to the test set. By contrast, QP-MIRA must be trained over a set of gold standard dependency trees to learn dependency weights prior to being tested. Cycle-forbidding threshold and solver timeout value were chosen based on a grid search over the dev set reordering results.

Data Penn treebank constituency-based trees were converted to dependency trees (Marcus et al., 1994) using the LTH conversion tool (Johansson and Nugues, 2007) to serve as gold trees. Trees in sections 2–21 of this converted treebank were used for the training purpose and sections 23 and 24 for test and development set, respectively.

Baseline We compare our work with two baselines. The first baseline is the CLE system (Wan et al., 2009). This work serves as a benchmark for the bare ILP system, no linguistic constraint, as both are using an MST approach to assign a dependency tree to a bag-of-words. The second baseline is the Assignment-based (AB) system (Wan et al., 2009). This system extends the CLE system by introducing argument

Model Name	Word Valency Constraints			
	PUNC	COORD	PREP	NOUN-VERB
M1	0	0	0	0
M2	0	0	0	1
M3	0	0	1	0
M4	0	0	1	1
M5	0	1	0	0
M6	0	1	0	1
M7	0	1	1	0
M8	0	1	1	1
M9	1	0	0	0
M10	1	0	0	1
M11	1	0	1	0
M12	1	0	1	1
M13	1	1	0	0
M14	1	1	0	1
M15	1	1	1	0
M16	1	1	1	1

Table 6.6: List of model names; presence of each constraint is indicated by 1

valency at dependency assignment time. Considering the intention of the linguistic constraints in the ILP and QP methods, the AB system becomes an appropriate baseline for comparison. Both of these baseline algorithms were discussed in the Section 2.3.2 in depth.

In case of failure at any stage of the word-ordering process, the model under investigation would back off to either of these baseline algorithms.

6.6.2 Models

Given the four sets of valency constraints that are introduced in Section 5.6.1, there are sixteen possible ways to combine them. To study the effect of an individual or a group of constraints, each combination will be regarded as a model. These models are all listed in Table 6.6. The presence of a specific constraint in a model is indicated by its value in the corresponding row, i.e. 1 means the constraint is active and 0 inactive.

We will evaluate ILP and QP with all the combinations (M1-M16) to investigate the effects of individual constraint types. For QP-MIRA we look just at M1, M15 and M16: that is, the version with no linguistic constraints, the version with the hardcoded linguistic constraints (i.e. without noun-verb valency), and the version

with all linguistic structure included. We will again specify which constraint sets have been used as we discuss each model's results.

6.6.3 Evaluation Metrics

Following the previous work reviewed in Section 2.3.2, we took BLEU¹ to measure the similarity between each original sentence and its re-generated alternative. This score is defined as the product of the n-gram precision and the brevity penalty. The n-gram is calculated for n ranging from 1 to 4. The brevity factor penalises the n-gram score if the target sentence length doesn't match the source sentence length, and is 1 otherwise. The more words are missed, the larger the penalty would be.

Assigning dependency trees to long sentences tends to be harder. Consequently, a higher rate of failure is expected in the process of reordering long sentences.

Through the following example, we demonstrate the relationship between a system's BLEU score and the length of the missing sentence(s). This has consequences for the use of the BLEU metric in comparing systems, as we illustrate below.

Figure 6.10 is an eight sentence sample of the ILP system output, T' , aligned with their source sentences, T . In Table 6.7, the BLEU scores of this ILP system along with six hypothetical systems (H1-H6) are listed for comparison. Each hypothetical system is able to regenerate all the given sentences except in a few cases, marked with BLEU score 0. H1, H2, and H3 failed in re-generating T_1 , T_6 , and T_8 respectively; all the three sentences have the same BLEU score of 0.41 when successfully re-generated ($\text{BlueScore}(T'_1) = \text{BlueScore}(T'_6) = \text{BlueScore}(T'_8) = 0.41$). However, the overall BLEU score of these systems differ. H2 has the lowest overall score, as it misses the longest sentence amongst the three. The same argument is valid for the high score of the H3 as it misses the shortest sentence. H6 describes a system that has a tendency towards regenerating short sentences. Regardless of the acceptable individual score for successful sentences, the overall score is rather low, due to the length of the missing sentences². We prefer to experiment on a generic model that covers sentences of arbitrary length. In other words, H6's overall score is so low that there is no indication of its success in re-generating the three of the eight of the input sentences with the score of 0.41.

Consequently, we report the quality of each system by two numbers.

¹The first surface realisation task (Belz et al., 2011) was carried out around the time this part of research was conducted. They used BLEU, NIST, METEOR and TER. The authors made no firm conclusion about superiority of these metrics but by just looking at results (see Table 6 in Belz et al. (2011)), they seems to give the same rankings. Therefore, there is no advantage or complementarity to use more than one.

²This can be regarded as a positive point for H6, if the purpose is to rank short sentences.

T_1 **no , it was n't black monday .**

T'_1 , was n't black monday no it

T_2 **but while the new york stock exchange did n't fall apart friday as the dow jones industrial average plunged 190.58 points - most of it in the final hour - it barely managed to stay this side of chaos .**

T'_2 this side of the dow jones industrial average plunged the new york stock exchange it the final hour to but in - n't as while most did friday 190.58 points fall apart stay managed barely chaos

T_3 **some ' circuit breakers ' installed after the october 1987 crash failed their first test , traders say , unable to cool the selling panic in both stocks and futures .**

T'_3 stocks circuit breakers , and to ' some traders say in after the selling panic both futures failed their first test unable installed cool the october 1987 crash

T_4 **the 49 stock specialist firms on the big board floor - the buyers and sellers of last resort who were criticized after the 1987 crash - once again could n't handle the selling pressure .**

T'_4 criticized the big board floor of last resort and the selling pressure on - n't were who the 49 stock specialist firms after could handle once again the 1987 crash sellers the buyers

T_5 **big investment banks refused to step up to the plate to support the beleaguered floor traders by buying big blocks of stock , traders say .**

T'_5 , to of by big blocks up big investment banks stock traders say buying support step refused the plate

T_6 **heavy selling of standard & poor 's 500-stock index futures in chicago relentlessly beat stocks downward .**

T'_6 & of stocks in chicago selling standard heavy downward beat relentlessly poor 's 500-stock index futures

T_7 **seven big board stocks - ual , amr , bankamerica , walt disney , capital cities div abc , philip morris and pacific telesis group - stopped trading and never resumed .**

T'_7 , walt disney philip morris and pacific telesis group - seven big board stocks trading never ual resumed stopped amr bankamerica capital cities div abc has

T_8 **the finger-pointing has already begun .**

T'_8 has already begun the finger-pointing

Figure 6.10: A subset of test set sentences along with their reordered alternative generated by ILP and QP systems. These sentences will be used to show the brevity penalty effect on the total BLEU a subset with missing sentences.

System	overall BLEU	BLEU score per sentence							
		T'_1	T'_2	T'_3	T'_4	T'_5	T'_6	T'_7	T'_8
ILP	0.31	0.41	0.32	0.26	0.31	0.10	0.41	0.39	0.41
H1	0.29	0	0.32	0.26	0.31	0.10	0.41	0.39	0.41
H2	0.26	0.41	0.32	0.26	0.31	0.10	0	0.39	0.41
H3	0.30	0.41	0.32	0.26	0.31	0.10	0.41	0.39	0
H4	0.23	0	0.32	0.26	0.31	0.10	0	0.39	0
H5	0.23	0.41	0	0.26	0.31	0.10	0.41	0.39	0.41
H6	0.0042	0.41	0	0	0	0	0.41	0	0.41

Table 6.7: Comparison of the BLEU score for the six hypothetical systems

1. The coverage percentage, $\frac{cnt(T_s)}{cnt(T)}\%$, the ratio of successfully generated sentences T_s to the whole test set T ;
2. BLEU score calculated over the set T'_s .

Here, T is the test set and $T_s \subset T$ is defined as a set of sentences that were successfully re-generated using the proposed algorithm. We will refer to such re-generated sentences as T'_s . T_f , the set of failed sentences, is defined as $T - T_s$. In other words, T_f is the set of test sentence that the proposed algorithm failed to linearise, either because the cycle limit or the time limit was exceeded during the assignment of a tree to the bag-of-words.

Each model within each system has a unique T_s . Therefore, we applied baseline algorithms to each T_s and calculated BLEU score over T'_s to provide an evaluation of those re-generated sentences for which a tree could be generated. These scores are listed in all the result tables under the **passed** (T'_s) header. We also provided the BLEU score for the baseline algorithms over T_f under the header **failed** (T'_f). Finally, the overall BLEU score per model reports the BLEU score of $T' = T'_s + T'_f$, where T'_s is the set of reordered sentences using the proposed constraints and algorithm, and T'_f is the set of reordered sentences produced by the specified baseline algorithm. Assuming H4 backs off to either CLE or AB, T'_s and T'_f would be as follows:

$$T'_s = \{T'_2, T'_3, T'_4, T'_5, T'_7\}$$

$$T'_f = \{T'_1, T'_6, T'_8\}$$

For all the introduced systems, all the above scores will be provided in a table followed by two diagrams that compare models against the baselines over T'_s and T' respectively.

Coverage percentage is the other indicator of each model confidence. To study

the correlation between the size of a bag-of-words and the presence of various constraints, we group the sentence length into the five following buckets:

- SS (short-sentence) for 1–10 words;
- MS (medium-sentence) for 10–20 words;
- LS (long-sentence) for 20–25 words;
- VLS (very long-sentence) for 25–30 words; and
- XLS (extremely long-sentence) for 30+ words.

The coverage percentages are then broken down into these buckets for each Model. We use stacked bar charts to provide a visual comparison of the competency of each model in handling bag-of-words with various length. Each bucket is represented by a unique color in those bar charts.

As a final note, to keep consistency with the previous experiments and make the results comparable, the minimum alterations were applied to each module. For example, the learning algorithm, MIRA, is adapted from the released³ implementation of dependency parsing (McDonald et al., 2005a,b).

Also, for the linearisation, we used the LM-based linearisation module that Wan et al. (2009) developed for their work. This module assigns the final word sequence by traversing the MST using a greedy edge selection algorithm. For nodes with more than one dependent, an n-gram language model score is calculated over all the possible combinations of those sibling nodes, and then the one with the highest score would be chosen as the best sequence to represent the given set of siblings.

We used Lpsolve and Gurobi libraries as the optimisation problem solver in the ILP and QP systems, respectively.

6.7 Evaluation Results

6.7.1 ILP

All the models including ILP consume the bag-of-words version of test sentences and assign each a dependency tree whose nodes are partially ordered using the ILP method with various combinations of valency constraints. These dependency trees are then linearised to produce the output sentences, T'_s . The initial objective of using the ILP method was its flexibility to add new constraints conveniently. However, the competency of the ILP system must be proved by answering the **RQ6.1** and **RQ6.2**.

³<http://www.seas.upenn.edu/~strctlrn/MSTParser/MSTParser.html>

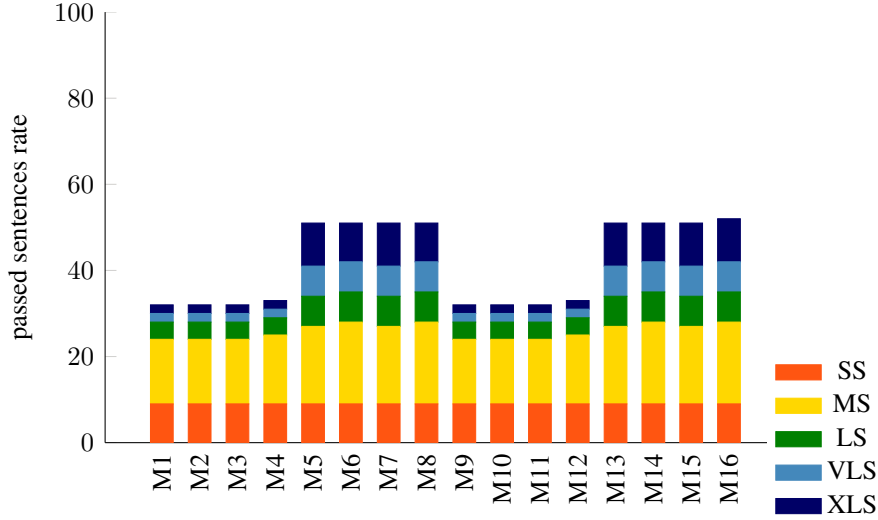


Figure 6.11: ILP coverage rate broken down by the length bucket: SS: 1–10 words MS: 10–20 words, LS: 20–25 words, VLS: 25–30 words, XLS: 30+ words.

ILP with no constraints vs CLE To answer **RQ6.1** we need to look at Table 6.8 and the M1 row which specifies the model with no linguistically-motivated constraints i.e. the equivalent to the CLE. Comparing their BLEU% over the T_s shows that the ILP tends to beat both baselines. This behaviour can be seen consistently with all the other models (M2–M15). We attribute this to the fact that there are not necessarily one unique MST per graph and the idiosyncrasies of these two algorithms, e.g. their cycle removal process, lead each algorithm to a different tree. According to the BLEU%, the trees that are produced by ILP tend to have a better linearisation.

Contribution of constraints To study the effect of linguistically motivated constraints, we need to compare M1 with M2–M16 in Table 6.8. Comparing M1’s coverage that has no valency constraint with either of M2, M3 and M5, it is apparent that the presence of the NOUN-VERB-ARITY, PREP-VALENCY, and COORD-VALENCY constraints contribute to coverage percentage. On the other hand, PUNC-VALENCY constraint has no steady influence on the coverage: Compare M1 with M9 (no change), M2 with M10 (decrease) and M8 and M16 (increase). As each group of the valency constraints targets a specific class of words, it is expected that a higher coverage is gained by applying a combination of positively contributing constraints. Interestingly, the combination of NOUN-VERB-ARITY and PREP-VALENCY in absence of COORD-VALENCY always improves the coverage compared to each alone. In other words, M4’s coverage is better than either M2 or M3; the same observation can be made for M12 when compared with M10 and M11.

model	cov.	passed-only			failed-only		back-off	
		ILP	CLE	AB	CLE	AB	CLE	AB
M1	31.49%	40.68	31.18	33.48	29.06	32.37	31.49	34.08
M2	31.83%	39.31	31.44	33.57	29.07	32.45	31.27	33.88
M3	31.79%	42.66	31.42	33.34	29.02	32.43	31.82	34.59
M4	31.92%	41.98	31.75	33.36	29.01	32.48	31.63	34.53
M5	51.32%	34.21	28.64	30.78	29.80	33.14	31.81	33.62
M6	50.55%	33.33	28.80	31.15	29.87	33.21	31.79	33.26
M7	51.32%	35.74	28.58	30.80	29.84	33.20	32.54	34.34
M8	50.64%	35.19	28.85	31.09	29.84	33.24	32.07	34.13
M9	31.49%	40.81	31.37	33.40	29.01	32.39	31.48	34.12
M10	31.62%	39.26	31.44	33.59	29.09	32.46	31.29	33.87
M11	31.75%	42.74	31.54	33.46	28.99	32.44	31.96	34.62
M12	31.79%	42.16	31.87	33.65	28.98	32.50	31.91	34.57
M13	51.36%	34.27	28.73	30.76	29.74	33.15	31.80	33.65
M14	50.51%	33.30	28.80	31.13	29.87	33.22	31.81	33.26
M15	51.32%	35.79	28.67	30.80	29.77	33.19	32.52	34.37
M16	50.72%	35.33	28.88	30.96	29.76	33.27	32.11	34.21

Table 6.8: The ILP evaluation result: coverage and BLEU% over T_s and T along with the corresponding CLE and AB scores. The back-off column is over the whole set T and represents the ILP system backing off to CLE or AB in case of failure.

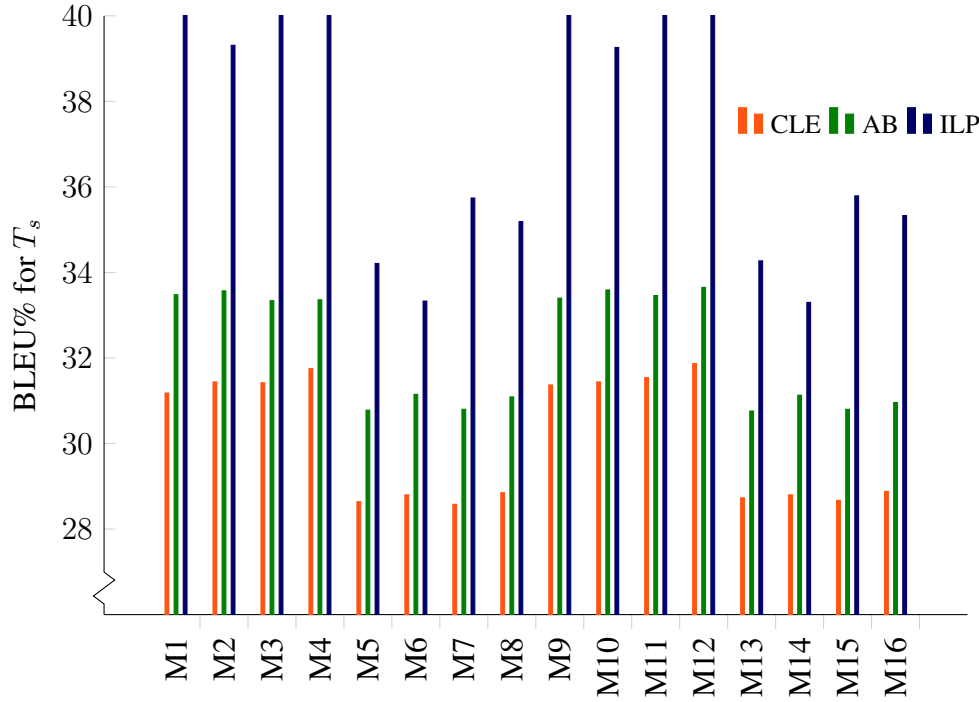


Figure 6.12: Comparison of the ILP system and the baseline algorithms over T_s .

A further observation is that COORD-VALENCY has the highest contribution to the coverage percentage, M5. As Coordination conjunctions are more likely to be present in long sentences, the expectation is that the increase is due to ability of this constraint to handle long sentences. A more clear picture of coverage percentage is provided in Figure 6.11. Each color in each bar represents the previously introduced length buckets (see page 161). This figure supports the above mentioned hypothesis; COORD-VALENCY constraints specifically assists in assigning dependency tree to sentences that are marked as long, very long, and extremely long.

Figure 6.12 compares these three systems. It is apparent that all the ILP models perform better than both baselines over their own specific T_s . It is worthwhile to remind the reader that in this stage T_s for each model is different. Therefore, each model can only be compared with its baselines, and not with one another. The best model, the one with the largest difference to its baselines, is M3. This enhancement suggests the PREP-VALENCY constraint creates better dependency trees whose nodes are partially ordered. ILP models based on M5–M8 and M13–M16 suffer from a noticeable drop in the BLEU score over T_s , compared to M1–M4 and M8–M12. Similar behaviour can be seen with their corresponding baselines but with a slighter degree. This suggests the difficulty in ordering long sentences.

Figure 6.13 compares the BLEU scores of all these system over the whole test set

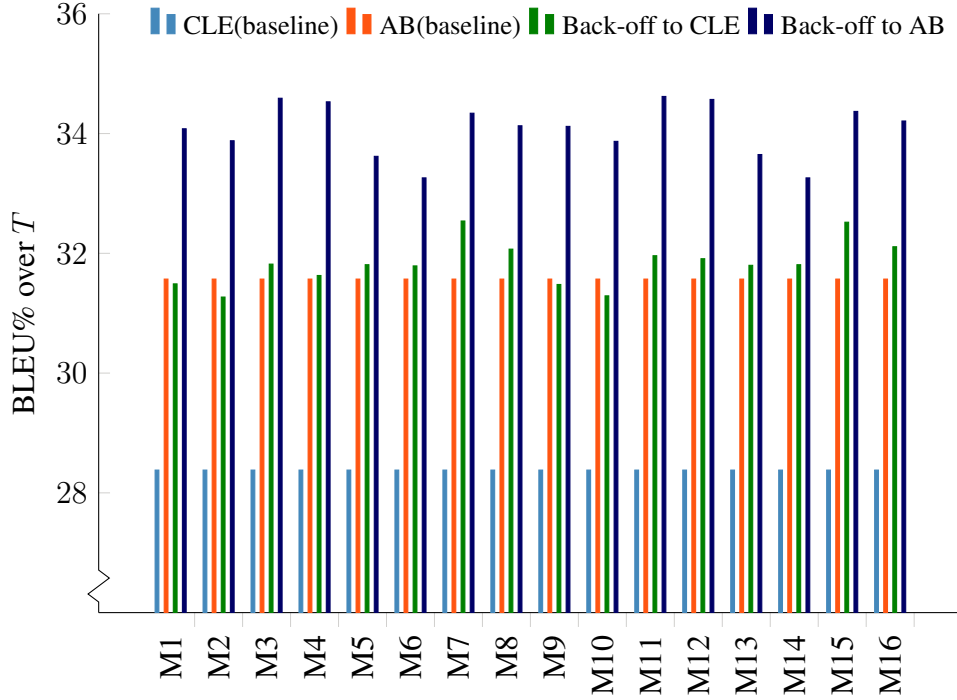


Figure 6.13: Overall comparison of the ILP system and the baseline algorithms.

T ; ILP backed-off to one of the baselines when it failed. This figure shows the superiority of ILP over the baselines. In other words, those 30%–50% of sentences that were re-generated by ILP caused this enhancement in BLEU score. A slight drop in the BLEU score is observed when NOUN-VERB-ARITY is added to any model; Compare $M(2N+1)$ to $M(2N+2)$ for $0 \leq N \leq 7$. This drop indicates the way NOUN-VERB-ARITY constraints are defined is not contributing to the BLEU score. The QP System that employs a different definition of NOUN-VERB-ARITY constraint is the next model to be evaluated.

6.7.2 QP

Following the discussion in Section 5.6.2 and the evaluation results of the ILP system in the previous section, NOUN-VERB constraints are not guiding the search as well as might be hoped. In the QP system, such constraints were redefined and embedded into the objective function as the penalty terms in the form of a quadratic function. It is essential to evaluate QP to establish the effect of this alteration in dependency tree creation and re-generating task. In the QP system, models with NOUN-VERB are those who have the penalty term in the objective function activated, and the rest has no such constraint at all.

All the experimental results for the QP along with their corresponding baseline

model	cov.	passed-only			failed-only		back-off	
		QP	CLE	AB	CLE	AB	CLE	AB
M1	36.54%	40.47	30.56	33.42	27.48	32.33	31.73	34.14
M2	96.58%	30.21	29.32	32.38	32.03	0.71	30.28	30.11
M3	35.82%	43.25	31.10	33.93	27.24	32.25	32.35	34.79
M4	96.62%	30.28	29.34	32.38	28.37	0.65	30.34	30.17
M5	54.05%	34.46	28.72	31.54	27.90	32.94	32.00	33.64
M6	96.58%	30.21	29.32	32.38	32.03	0.71	30.28	30.11
M7	53.42%	36.93	29.07	31.72	27.60	32.88	32.99	34.75
M8	96.62%	30.26	29.33	32.37	28.37	0.65	30.32	30.15
M9	32.78%	40.28	31.36	34.18	27.26	32.25	31.50	33.95
M10	96.58%	30.19	29.32	32.38	32.03	0.71	30.26	30.10
M11	33.71%	43.23	31.85	34.21	27.12	32.23	32.21	34.70
M12	96.62%	29.51	29.13	32.78	19.49	1.52	29.56	29.45
M13	51.98%	34.38	28.72	31.39	27.80	33.04	31.91	33.64
M14	96.58%	30.19	29.32	32.38	32.03	0.71	30.26	30.09
M15	52.45%	36.86	29.04	31.48	27.60	32.97	32.96	34.7
M16	96.62%	30.24	29.33	32.37	28.37	0.65	30.30	30.13

Table 6.9: Coverage and BLUE score of QP over T_s as well as the over all BLEU score along with the baseline algorithms' scores over the corresponding test sets. The back-off column is over the whole set T and represents the QP system backing off to CLE or AB in case of failure.

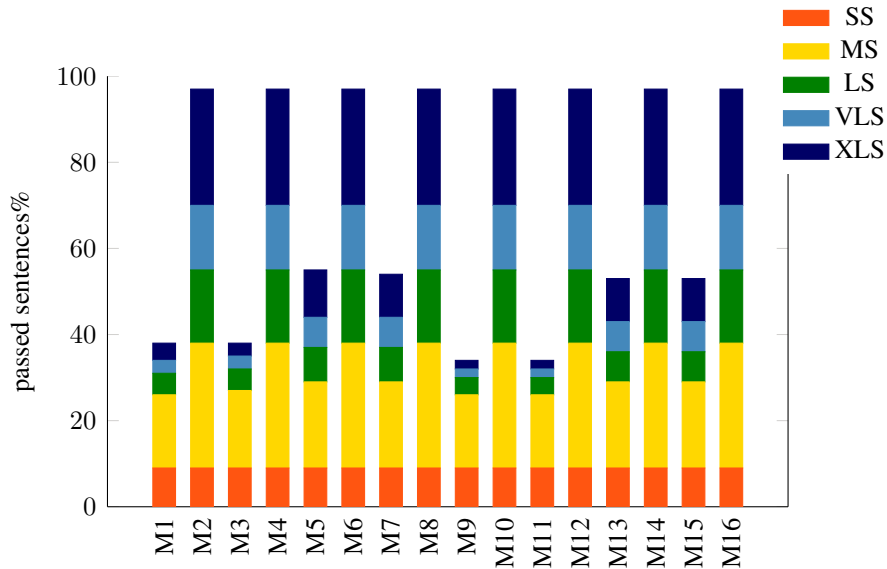


Figure 6.14: QP coverage rate broken down to length bucket: SS: 1–10 words MS: 10–20 words, LS: 20–25 words, VLS: 25–30 words, XLS: 30+ words.

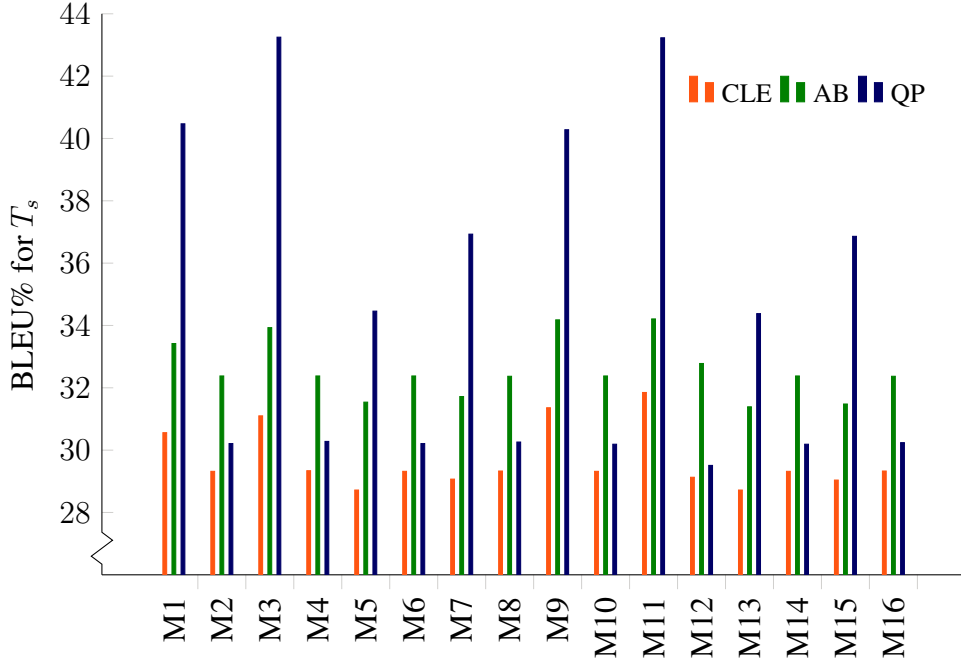


Figure 6.15: Comparison of the QP system and the baseline algorithms over T_s .

scores are listed in Table 6.9. Figure 6.14 demonstrates the relationship between the presence of the penalty term and the coverage percentage. The high coverage rate for models ending with even number implies that NOUN-VERB-ARITY are more beneficial in successful dependency tree creation if presented as penalty term in the objective function than being applied as hard constraints.

Figure 6.15 compares the quality of the QP reordered sentences with the baseline algorithms. Similar to the COORD-VALENCY case in the ILP system, those long sentences that were covered by NOUN-VERB-ARITY cause a drop in BLEU score so that those models can only beat the CLE baseline but not the AB baseline.

The high coverage of models with the NOUN-VERB-ARITY constraint, about 96.5%, makes backing-off to baseline ineffective, regardless of which algorithm it backs-off to. This explains why backing-off to CLE and AB for such models score almost the same in Figure 6.16. This figure compares the overall BLEU scores of the QP system and the two baseline algorithms. As expected, models with no NOUN-VERB-ARITY beat both of the baselines. Models with NOUN-VERB-ARITY as a penalty term are still superior to the CLE baseline.

In conclusion, the relatively low BLEU score for the NOUN-VERB models suggest reviewing the dependency selection parameters. The most important dependency selection parameter is dependency weights. Consequently, the next model's target is to find the best weights for dependency.

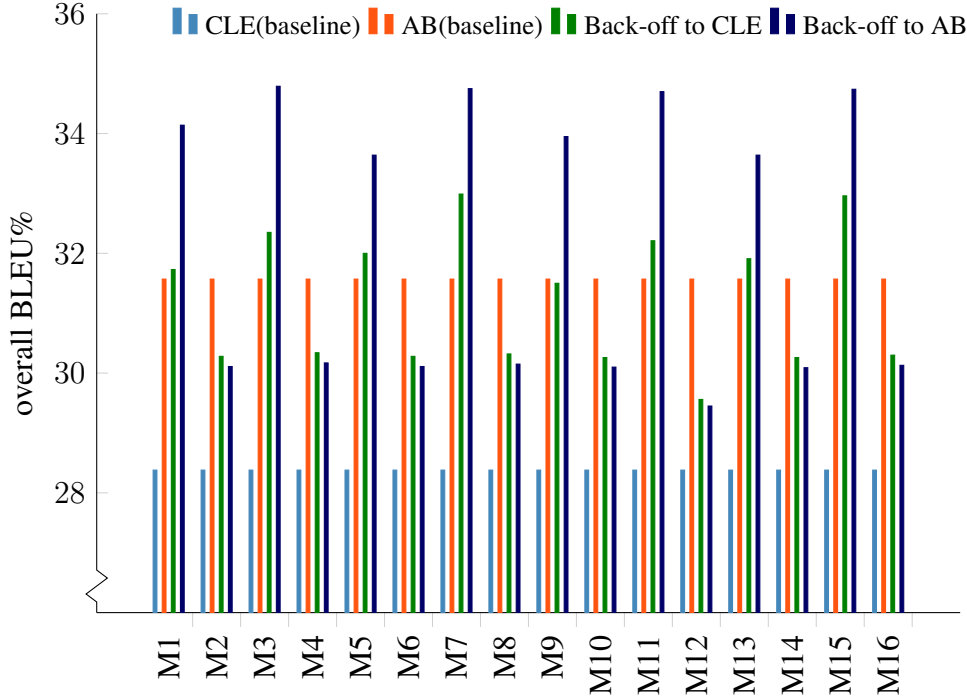


Figure 6.16: Comparison of the QP system and the baseline algorithms over T_s .

6.7.3 QP-MIRA

The two previously discussed models employ exactly the same weights as CLE and AB algorithms. In this section, the goal is to analyse the quality of reordering performed by a system that learns the dependency weights from a treebank using the MIRA learning algorithm (see Section 6.1).

To study the effect of different parameters in learning, the QP-MIRA system is trained and tested over seven new models. These models are defined in Table 6.10 based on M1, M15, and M16, which were previously introduced in Table 6.6; we choose these as the basis for the seven new models for the following reasons.

Our QP models allow more flexibility as we move from hard to soft constraints, by embedding NOUN-VERB-ARITY to objective function as penalty terms. However, it seems that the weights from Wan et al. (2009) are not effective anymore. So provided MIRA proves to be helpful in learning weights for this scenario, it means it can learn dependency weights in case of less complicated models such as our ILP models and also for languages that do not have a dependency treebank large enough to extract weights e.g., using dependencies' log-probability.

Our selected models for the QP-MIRA system were M1, M15 and M16. From the previous section, M15 (all constraints less noun-verb arity) is one of the two top models, and M16 is M15 with the noun-verb arity constraint — the one that has

model	# training iterations	cycle-forbidding constraints threshold	ILP/QP model
MIRA1	1	50	M1
MIRA2	1	50	M15
MIRA3	2	50	M15
MIRA4	1	100	M16
MIRA5	2	100	M16
MIRA6	5	100	M16
MIRA7	1	350	M16

Table 6.10: QP_MIRA system is trained and tested over MIRA1–MIRA7 models which are based on three ILP models M1, M15 and M16 (discussed in the previous section) and differ in the number of iterations used for training MIRA (learning weights) and the threshold used for controlling the incremental generation of cycle-forbidding constraints.

model	cov.%	passed-only			failed-only		back-off	
		QPMIRA	CLE	AB	CLE	AB	CLE	AB
MIRA1	82	36.38	28.25	31.25	24.18	26.43	34.02	34.45
MIRA2	84	38.80	27.98	31.57	24.55	26.55	36.43	36.84
MIRA3	71	38.51	28.54	31.41	24.89	27.48	34.37	35.26
MIRA4	81	37.69	28.25	31.61	24.00	26.69	34.99	35.66
MIRA5	63	36.85	28.30	31.41	26.08	28.64	32.83	33.81
MIRA6	42	38.40	29.03	31.82	26.00	29.01	30.36	32.24
MIRA7	84	37.13	28.14	31.73	24.43	25.96	34.92	35.24

Table 6.11: Coverage and BLUE score of QP over T_s as well as over T , along with the baseline algorithms score over the corresponding test sets.

not had any positive impact so far — included. M1 is the usual baseline with no linguistic constraints.

Similar to previously discussed systems, the evaluation results are shown in Table 6.11 followed by the corresponding comparative graphs in Figure 6.17 and Figure 6.18.

Since the coverage percentages have been improved and MIRA modifies the initial dependency weights, we would like to see how close the produced dependency trees are to the Gold standard trees. Figure 6.17(a) compares the quality of the generated dependency trees using QP-MIRA. MIRA2, MIRA4 and MIRA7 have the closest proportion of generated trees to the gold standard trees, amongst all (Figure 6.17(b)); they have the lowest failure rate and the precision of the most of the generated trees, T'_s , fell in bucket 25%–50%. Also these models have the largest share of the two top precision buckets.

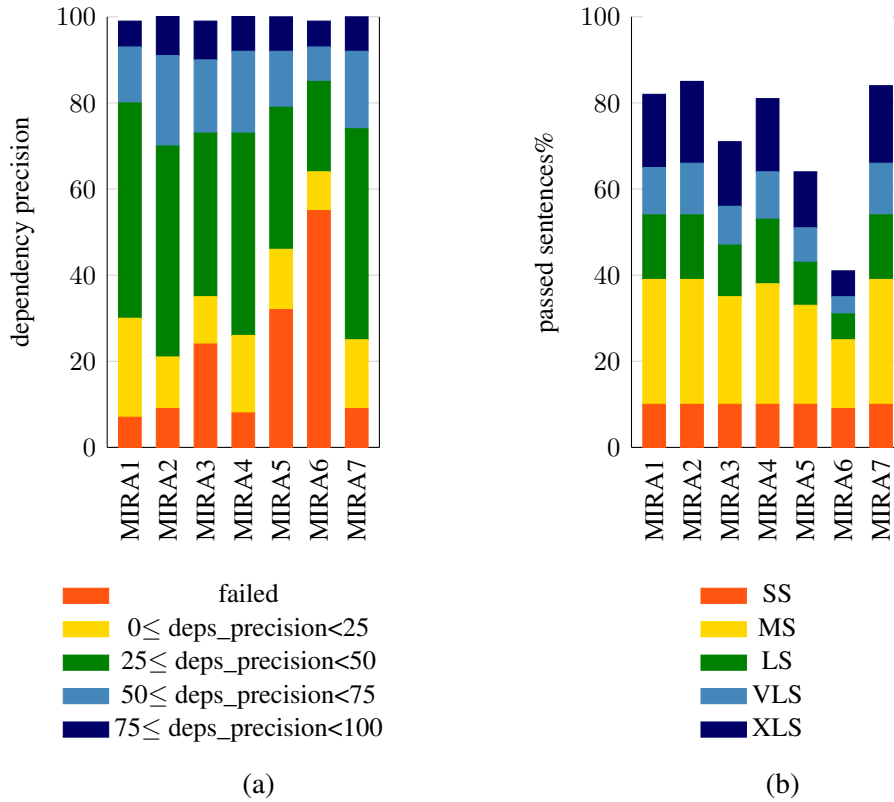


Figure 6.17: (a) Quality of generated dependency trees using QP-MIRA; (b) QP-MIRA Coverage rate broken down to length bucket: SS: 1–10 words MS: 10–20 words, LS: 20–25 words, VLS: 25–30 words, XLS: 30+ words.

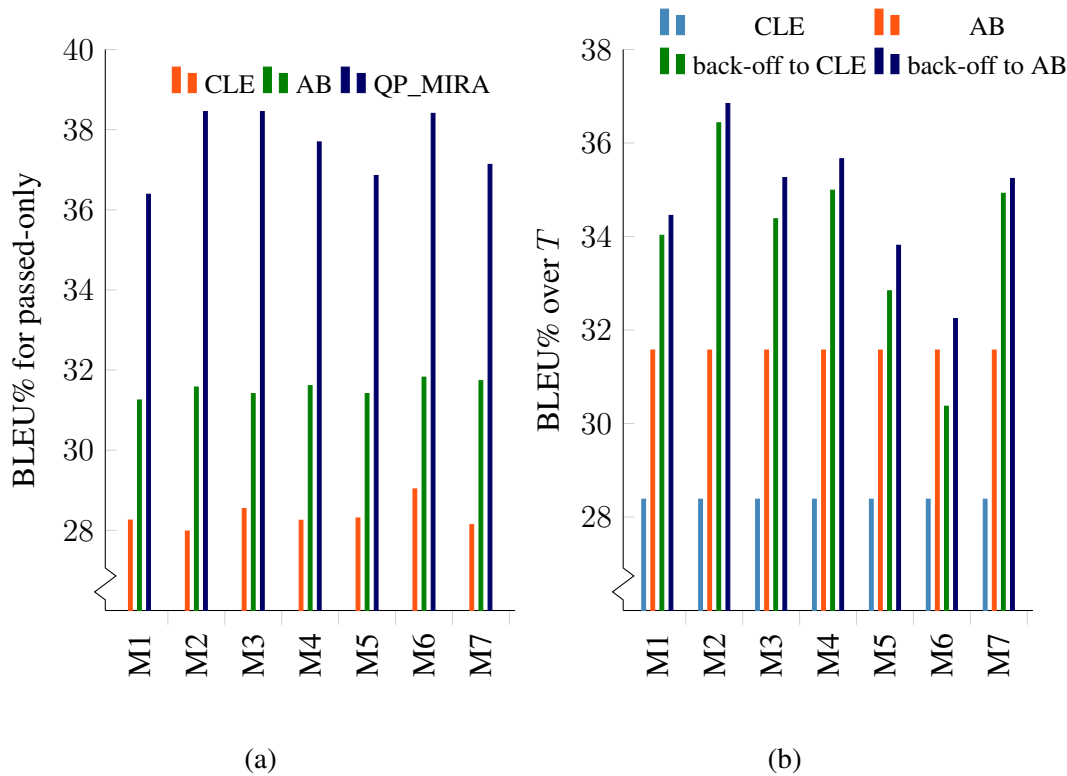


Figure 6.18: QP-MIRA compared to its baseline algorithm (a) over T_s and (b) over the whole test set T .

The two bar charts in Figure 6.18 compare the quality of these models over T_s and T respectively. The results in Figure 6.18(b) confirms that QP-MIRA beats both baseline for both test sets by a large margin. Unlike the two previous systems that compromise between the coverage and quality of the generated sentences, this system provides high coverage and quality sentences at the same time. These improvements confirm that learning dependency weights enhances the result of the regeneration task. However, the gradual drop in the quality of dependencies, coverage, and BLEU score for higher numbers of training iterations suggests either overfitting or the training is not done against quite the right factors. This issue requires further study.

Figures 6.19(a) and Figure 6.19(b) compare M1 (aka MIRA1), M15 (aka MIRA2) and M16 (aka MIRA4) for ILP, QP and QP-MIRA. It can be clearly concluded that QP-MIRA not only retains the high coverage but also provides a significant increase in the BLEU score; compare QP and QP-MIRA in M16. However, comparing QP-MIRA for M15 and M16 shows that NOUN-VERB-ARITY even as a penalty term does not enhance the BLEU score.

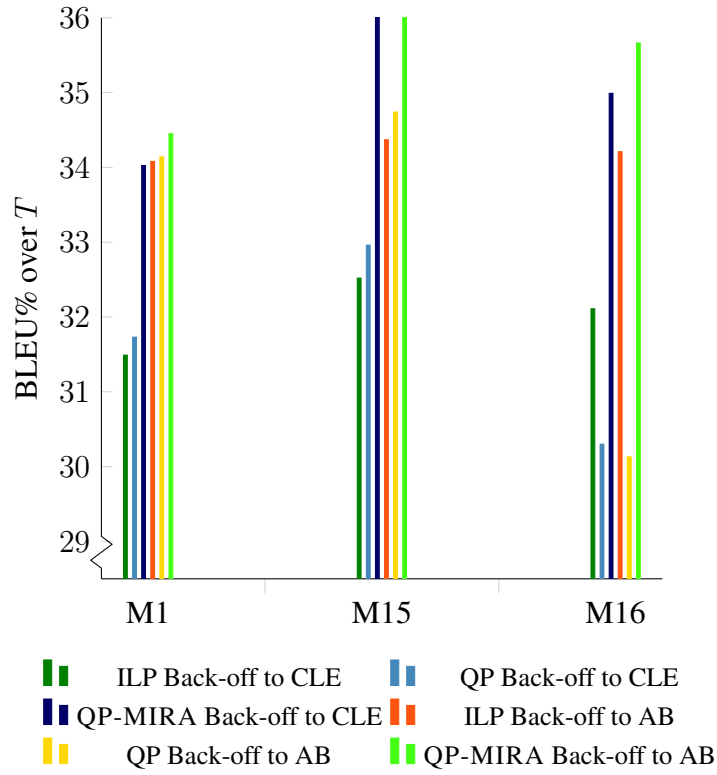
6.8 Comparison and Conclusion

The main goal of the current study was to answer the following three questions:

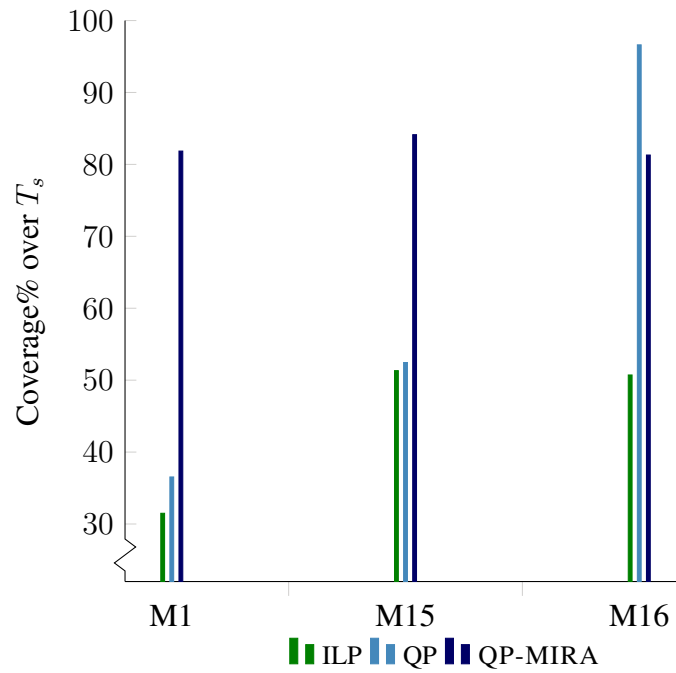
- Are mathematical optimisation methods such as ILP able to do MST identification as effectively as the CLE algorithm?
- Given some linguistic constraints applied to the MST problem, how much do such constraints contribute to the quality of the linearised string?
- Do learning dependency weights improve the output of the string regeneration?

The first question can be answered by looking at the M1 column in Figure 6.13. M1 has only tree constraints. It can be seen that the ILP beats both of the baselines it backs-off to. Unlike AB, CLE and ILP assign an MST to a string in the absence of any linguistic preferences. Both algorithms use the same set of weights, but as there is not just one unique MST for many of the input strings, each system ends up with a different set of dependency trees. One interpretation for this improvement in the BLEU% could be that ILP tends to pick trees that are linguistically more plausible. Therefore, it can be concluded that string regeneration can benefit from mathematical approaches in the MST creation phase.

To answer the rest of the research questions, we compare the systems against one another. Figure 6.20 compares the BLEU% of the ILP and the QP systems over the whole test set. In theory, the QP models with no NOUN-VERB-ARITY are



(a)



(b)

Figure 6.19: Comparison of ILP, QP and QP-MIRA when tested with three different constraint sets in terms of their (a) BLUE score over T when backed off to the baseline models, and (b) coverage% over T_s . M1 is the model with tree constraints only; M15 has tree constraints plus PUNC-VALENCY, COORD-VALENCY and PREP-VALENCY constraints; M16 has NOUN-VERB-ARITY in addition to all the M15 constraints.

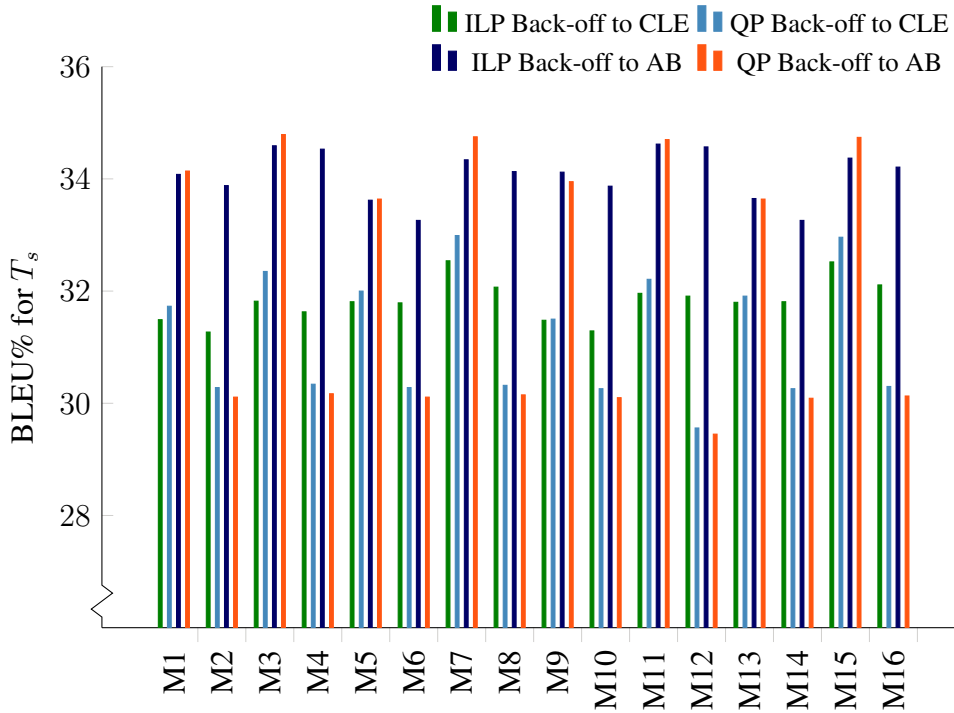


Figure 6.20: Comparison of the ILP and QP over the test set, T .

identical to their corresponding models in the ILP system. One interpretation of slight difference in the BLEU score can be due to the individual solvers' choices at the tree building phase.

Following the comprehensive comparison of the linguistic constraints in Section 6.7.1, and the BLEU% of M1 and M15 in Figure 6.19(a), it can be concluded that linguistic constraints contributing to the quality of re-generated strings significantly. As NOUN-VERB-ARITY constraint does not contribute to the ILP in the expected way, they are converted into penalty terms in QP. This switch causes a noticeable increase in coverage. However, this high coverage has a negative impact on the BLEU score. The intention of QP-MIRA is to fix this problem by learning the dependency weights from scratch.

As the final conclusion, QP-MIRA is superior to QP and ILP, regardless of the algorithm it backs off to. These findings further support the idea of training dependency weights for the sentence re-generation task, as for parsing. ILP and its related mathematical optimisation methods e.g. QP are useful approaches in string regeneration task using MST as their constraints provide a flexible tool to include extra information individually. This extra information changes the task into a partial word-ordering problem.

Improvement of the NOUN-VERB-ARITY constraints is one of the subjects of future work. The other possible future work could focus of the drop in the quality of the

Model	CLE	AB	ILP		QP		QP-MIRA	
	-	-	+CLE	+AB	+CLE	+AB	+CLE	+AB
BLEU	28.30	31.57	32.54	34.62	32.99	34.79	36.43	36.84

Table 6.12: Comparison of our baselines (CLE and AB) with the best of ILP, QP and QP-MIRA over the whole test set, T , where +CLE and +AB specify the back-off algorithms.

dependency trees, coverage%, and BLEU score with the increase in the number of training iterations.

The goal of these two chapters was to look at the effect of modelling the graph-based approach of Wan et al. using declarative constraints and ILP. Hence all the comparisons have been with respect to that approach (see Table 6.12). Nevertheless, we also comment on the results of Zhang et al. (2012).

Comparing the BLEU scores, Zhang et al. (2012) results are better than ours. Even, Zhang and Clark (2011) alone has a larger improvement (BLEU 40.1). However, a closer study shows that the improvements in Zhang et al. (2012) are for the most part via formalism-specific adjustments, e.g. in the algorithm for building derivations, or in tuning the supertagger for lexical pruning; not relevant to the graph-based approach. Other improvements come from improving the language model used. Consequently, we can conclude that their proposed approach which has led to fairly large improvements is orthogonal to the work we presented in this chapter.

Since we showed that it's helpful to use ILP and declarative constraints in the graph-based approach, it could possibly be complementary to their alternative approach. Potentially interesting to explore.

Chapter 7

Conclusion

7.1 Summary of Findings

The principal motivation of this thesis has been to investigate techniques for realisation in Natural Language Generation that will be useful in the context of minority languages — languages with a small grammar, a grammar in development phase or no computational grammar for generation — with surface realisation, specifically through two approaches: (1) grammar-based realisation ranking and (2) realisation from bag-of-words input. Previous studies addressed these two tasks with methods that heavily revolve around manually-crafted grammars and/or rich treebanks that allow grammar induction. Since these resources are expensive and time-consuming to obtain our aim was to identify methods and resources that could move surface realisers one step closer to a generic tool, similar to universal parsers.

In Chapter 3, we looked at various types of statistical parsers ranging from supervised PCFG parsers to unsupervised dependency parsers as potential resources for extracting grammatical features that could be an alternative to those extracted from symbolic grammars. The ranking models' accuracies gradually dropped as we moved towards the unsupervised parser but still, given the use of suitable feature selection criteria, structural features extracted from unsupervised parsers served as suitable source for incorporating grammaticality into the surface order. This becomes noteworthy if we compare the amount of effort required to build symbolic grammars, supervised parsers and unsupervised parsers. In terms of suitable feature selection criteria, we found the Information Gain was effective at incorporating unsupervised parser features. We also looked at reliability-based measures as another feature selection criteria which did not prove useful. We attribute its poor performance to systematic choices made by the unsupervised parser that were different from the gold standard's choices, rather than bad parsing. We also showed that the structural information captured by such features are somehow orthogonal

to the word order information provided by n-gram language models, as models that were built upon the combination of these features provided higher accuracies. The experimental results confirmed our speculations about the relationship between the size of the language model and its contribution to the accuracy: the larger the language model, the more accurate the ranking model is.

Following the contribution of features extracted from the unsupervised parser to the ranking realisation model, in Chapter 4, we looked into how such features work when compared against/combined with features extracted from generators internal representation. The latter set of features were previously proved to be helpful in ranking alternatives in German (Cahill et al., 2007b). Following Cahill et al. (2007b), we used an LFG-based text generator, so we looked at similar features for our generator’s internal representation features and built various models using different combinations of c-/f-structure features. Our experimental results with internal features were in line with the previous research in terms of the usefulness of such features in ranking the realiser’s output.

The accuracies of the hybrid models — combinations of statistical parser features with the system’s internal representation features — were, however, no better than models built upon features from the symbolic grammar. This demonstrates that our features from the statistical parsers were not capable of capturing more than what features from manually crafted grammars can capture. In other words, given our choices of parsers and grammar, these two resources are not complementary to one another.

We also looked at the effects of size of the grammar, by investigating the fragment LFG grammar that comes with XLE. Models with features from the small grammar tended to have higher accuracies over a smaller effective test set. However, further analysis revealed that the generated alternatives with the small grammar only differ in uninteresting ways. Thus, the contribution of the large and small grammars to realisation ranking is not comparable.

The second task that we tackled was based on a graph-based data-driven generation model that generates sentences from a bag-of-words input. Our proposed framework, introduced in Chapter 5, is based on the previous work of Wan et al. (2009). Under this framework we reduced the generation problem into finding the Minimum Spanning Tree (MST) as an intermediate structure from which word order can be inferred. The key idea in the framework of this thesis has been to replace the Chu-Liu-Edmonds (CLE) algorithm with an Integer Linear Programming (ILP)-based approach to benefit from the flexibility that comes with declarative constraints. Constraints provide a consistent way of defining and embedding structural and grammatical expectations, i.e. a connected, cycle-free structure with maximum weights that has limitation on the number of dependents each node can take, such that all of them can be satisfied at the same time. However, we found that some of the language-specific constraints like node arity may lead

to infeasibility. As a solution, we proposed a second version of the framework where we changed the single objective nature of the problem (finding the MST) into a multi-objective problem (finding the MST whilst keeping each node’s arity as close to gold standard valency as possible). For this purpose, we redefined arity constraints whose goal is to keep the number of each node’s dependents within hard-coded lower and upper bounds, to be penalty terms so the objective function is penalised proportional to the arity distance from the gold standard valency. Since we used a quadratic term to represent the above mentioned distance, the optimisation problem become a quadratic (QP) one rather than linear.

In pursuit of the original motivation of promoting algorithms and resources for minority languages, in Chapter 6, we attempted to learn dependency weights using the Margin Infused Relaxed Algorithm (MIRA) within the same framework rather than relying on the existence of a large dependency treebank for deriving MLE estimates of these dependency weights. Our experiment results showed that our novel ILP-based framework beats both baselines from Wan et al. (2009): the CLE one noted above, and their method for incorporating linguistic structure. Switching to QP in order to incorporate preferred word valencies had a noticeable increase in the coverage of sentences that the model can generate a structure for, but had a negative impact on the BLEU score. However, models that used MIRA not only maintained the high coverage but also showed superiority in terms of BLEU score compared to ILP and QP-based models.

7.2 Limitations and Future Outlook

The discoveries of the thesis have also led to a recognition of various limitations and caveats.

In this thesis, we attempted to replace or minimise the use of manually-crafted resources, such as symbolic grammars and treebanks to promote realisation and realisation ranking for languages that lack such resources or have them in development. One avenue for further exploration could involve looking at LFG grammars in development for other languages. These would be somewhere in size between the large English grammar and the fragment grammar, and it is possible the use of features derived from external features could be useful there.

As for realisation ranking, we suggest looking into other feature selection measures and other parser types such as transfer-based parsers. We also recommend future researchers to convert dependency trees to a universal format to avoid systematically different choices made by various parsers/grammars which influence the usefulness of the models; as we found out, the reliability-based feature selection method cannot be helpful with our existing set-up.

Intuitively, noun-verb-arity constraints seem to be a key type of constraint for useful linguistic tree structure. Our experimental results showed that neither of the implementations of such constraints improved results: ILP was too constraining, and QP wasn't constraining enough. One possible solution is to use the hard constraint formulation and incrementally relax constraints. Relaxation is a modelling strategy and provides an easy-to-solve approximation of a difficult problem; once ILP problem found to be infeasible, the hard constraints can be incrementally relaxed until the problem becomes feasible. The other possible approach is to find a better way of incorporating a penalty function in QP, for instance redefining the penalty function.

We used Google syntactic n-gram to guide our search for a reasonably grammatical tree which proved to be helpful when combined with the MIRA learning algorithm. As valuable it is, it is expensive to build such syntactic n-gram language model, regarding the gigantic corpora used in the project. As an alternative, we suggest development of an automatic or semi-automatic language-independent approach that can provide an estimation words' left and right arities using unsupervised parsers.

We discussed the recent popularity of deep learning in addressing the text generation problem in Chapter 2.4. In addition to suffering from almost no coherence, grammaticality of the generated sentences is another major issue with systems built upon deep learning methods. We propose a future study on incorporating the grammaticality constraints into Recurrent Neural Network (RNN)s to improve the produced text.

Appendix A

Small-Scale Trial

Weight matrices and their Maximum Spanning Tree (MST) generated by ILP (T') during the training phase are presented to enable the in-depth comparison of both models and also how the weights are changing from one iteration to the next.

Tables A.1-A.4 show weight matrices during the four-iteration training with FR-WO features, followed by their alternative MSTs in Figures A.1-A.4).

	root	range	and	Inc.	were	among	targets	.
root	0.000	0.318	-0.300	1.281	0.056	-0.364	-0.091	0.000
range	1.302	0.000	0.989	2.604	1.111	0.876	1.247	1.302
and	0.429	0.803	0.000	1.845	0.295	0.065	0.159	0.429
Inc.	0.851	1.121	0.646	0.000	0.665	0.483	0.760	0.851
were	0.709	1.183	0.435	2.102	0.000	0.405	0.564	0.946
among	-0.300	0.034	-0.600	1.077	-0.499	0.000	-0.385	-0.300
targets	0.838	1.269	0.449	2.085	0.647	0.458	0.000	0.838
.	0.000	0.318	-0.300	1.281	-0.186	-0.364	-0.091	0.000
root	0.000	0.512	0.300	0.851	0.670	0.364	0.375	0.000
range	0.931	0.000	1.207	1.793	1.174	1.353	1.309	0.931
and	1.046	1.684	0.000	2.167	1.380	1.410	1.062	1.046
Inc.	0.498	1.005	0.992	0.000	0.684	0.854	0.872	0.498
were	0.863	1.475	1.216	1.823	0.000	1.347	1.260	1.338
among	0.526	1.208	0.827	1.588	0.709	0.000	1.036	0.526
targets	0.315	0.774	0.534	1.134	0.491	0.698	0.000	0.315
.	0.000	0.512	0.300	0.851	0.186	0.364	0.375	0.000

Table A.1: Objective function (weight matrix) for sentence (6.1) in the first iteration of training with FR-WO features.

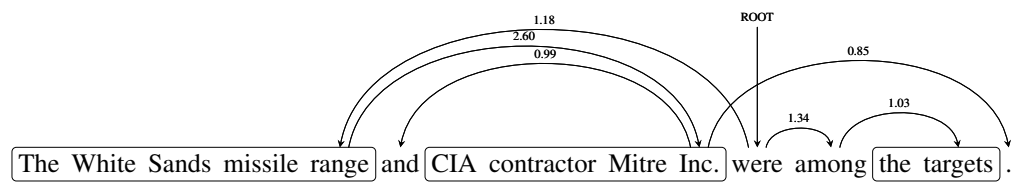


Figure A.1: ILP generated tree with initial weights. Arcs' labels are dependencies' weights

	root	range	and	Inc.	were	among	targets	.
root	0	0.473	-0.388	1.412	0.056	-0.390	-0.108	0
range	1.360	0	0.959	2.779	1.157	0.907	1.289	1.360
and	0.403	0.948	0	1.950	0.266	0.013	0.140	0.403
Inc.	1.048	1.480	0.756	0	0.855	0.654	0.939	1.048
were	0.669	1.297	0.286	2.233	0	0.289	0.508	0.922
among	-0.296	0.188	-0.684	1.234	-0.502	0	-0.408	-0.296
targets	1.176	1.896	0.712	2.561	0.978	0.770	0	1.176
.	0	0.473	-0.388	1.412	-0.192	-0.390	-0.108	0
root	0	0.630	0.388	0.940	0.690	0.390	0.419	0
range	0.833	0	1.196	1.777	1.075	1.278	1.256	0.833
and	1.076	1.866	0	2.286	1.423	1.466	1.185	1.076
Inc.	0.636	1.277	1.222	0	0.829	1.018	1.055	0.636
were	0.867	1.597	1.266	1.970	0	1.345	1.311	1.375
among	0.580	1.387	0.968	1.776	0.769	0	1.126	0.580
targets	0.464	1.108	0.801	1.371	0.647	0.875	0	0.464
.	0	0.630	0.388	0.940	0.192	0.390	0.419	0

Table A.2: Objective function (weight matrix) for sentence (6.1) in the second iteration of training with F-WO features.

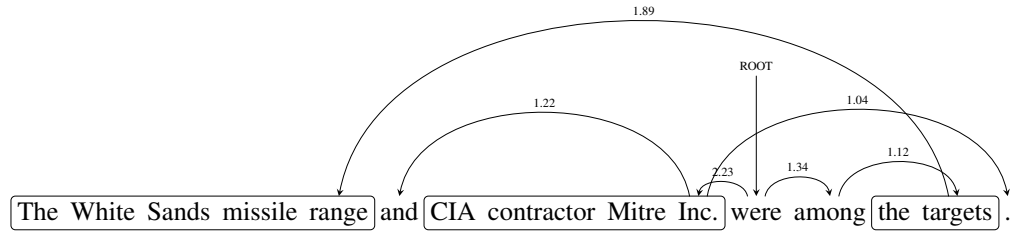


Figure A.2: ILP generated tree with initial weights. Arcs' labels are dependencies' weights

	root	range	and	Inc.	were	among	targets	.
root	0	0.473	-0.388	1.412	0.056	-0.390	-0.108	0
range	1.360	0	0.959	2.779	1.157	0.907	1.289	1.360
and	0.403	0.948	0	1.950	0.266	0.013	0.140	0.403
Inc.	1.048	1.480	0.756	0	0.855	0.654	0.939	1.048
were	0.669	1.297	0.286	2.233	0	0.289	0.508	0.922
among	-0.296	0.188	-0.684	1.234	-0.502	0	-0.408	-0.296
targets	1.176	1.896	0.712	2.561	0.978	0.770	0	1.176
.	0	0.473	-0.388	1.412	-0.192	-0.390	-0.108	0
root	0	0.630	0.388	0.940	0.690	0.390	0.419	0
range	0.833	0	1.196	1.777	1.075	1.278	1.256	0.833
and	1.076	1.866	0	2.286	1.423	1.466	1.185	1.076
Inc.	0.636	1.277	1.222	0	0.829	1.018	1.055	0.636
were	0.867	1.597	1.266	1.970	0	1.345	1.311	1.375
among	0.580	1.387	0.968	1.776	0.769	0	1.126	0.580
targets	0.464	1.108	0.801	1.371	0.647	0.875	0	0.464
.	0	0.630	0.388	0.940	0.192	0.390	0.419	0

Table A.3: Objective function (weight matrix) for sentence (6.1) in the third iteration of training with FR-WO features.

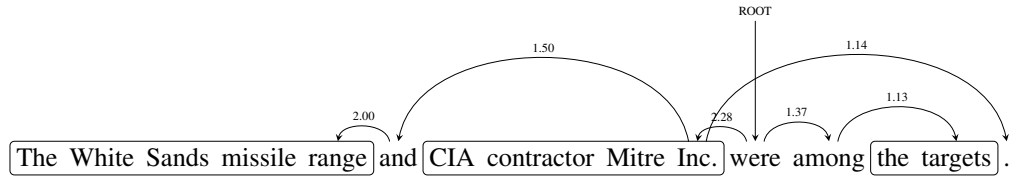


Figure A.3: ILP generated tree with initial weights. Arcs' labels are dependencies' weights.

	root	range	and	Inc.	were	among	targets	.
root	0.000	0.511	-0.474	1.501	0.056	-0.406	-0.113	0.000
range	1.469	0.000	0.981	2.982	1.266	1.004	1.393	1.469
and	0.479	1.063	0.000	2.115	0.342	0.073	0.211	0.479
Inc.	1.192	1.662	0.874	0.000	1.000	0.782	1.079	1.192
were	0.688	1.355	0.220	2.354	0.000	0.300	0.523	0.942
among	-0.299	0.239	-0.773	1.321	-0.504	0.000	-0.418	-0.299
targets	1.226	1.987	0.676	2.726	1.028	0.804	0.000	1.226
.	0.000	0.511	-0.474	1.501	-0.192	-0.406	-0.113	0.000
root	0.000	0.667	0.474	1.014	0.690	0.406	0.424	0.000
range	0.907	0.000	1.356	1.925	1.149	1.382	1.335	0.907
and	1.234	2.059	0.000	2.517	1.580	1.639	1.347	1.234
Inc.	0.775	1.452	1.564	0.000	0.967	1.173	1.199	0.775
were	0.894	1.661	1.379	2.095	0.000	1.392	1.343	1.402
among	0.601	1.474	1.075	1.872	0.790	0.000	1.149	0.601
targets	0.472	1.153	0.894	1.466	0.654	0.899	0.000	0.472
.	0.000	0.667	0.474	1.014	0.192	0.406	0.424	0.000

Table A.4: Objective function (weight matrix) for sentence (6.1) in the fourth iteration of training with FR-WO features.

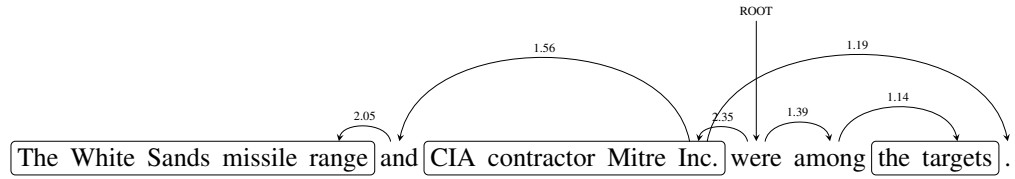


Figure A.4: ILP generated tree with initial weights. Arcs' labels are dependencies' weights.

Tables A.5-A.8 show weight matrices during the four-iteration training with F-WO features, followed by their alternative MSTs (Figures A.5-A.8).

	root	range	and	Inc.	were	among	targets	.
root	0	0	0	0	0	0	0	0
range	1.404	0	0	0	0	0	0	0
and	0.501	1.152	0	0	0	0	0	0
Inc.	1.014	1.604	0.871	0	0	0	0	0
were	1.062	1.731	0.808	2.797	0	0	0	0
among	0.055	0.639	-0.253	1.628	-0.059	0	0	0
targets	1.099	1.811	0.774	2.655	0.991	0.943	0	0
.	0	0.554	-0.308	1.535	-0.121	-0.146	0.078	0
root	0	0.782	0.308	1.352	0.481	0.146	0.326	0
range	0	0	1.356	2.413	1.145	1.181	1.320	1.040
and	0	0	0	2.808	1.417	1.272	1.277	1.126
Inc.	0	0	0	0	0.959	1.007	1.132	0.838
were	0	0	0	0	0	1.345	1.373	1.620
among	0	0	0	0	0	0	0.669	0.246
targets	0	0	0	0	0	0	0	0.774
.	0	0	0	0	0	0	0	0

Table A.5: Objective function (weight matrix) for sentence (6.1) in the first iteration of training with F-WO features.

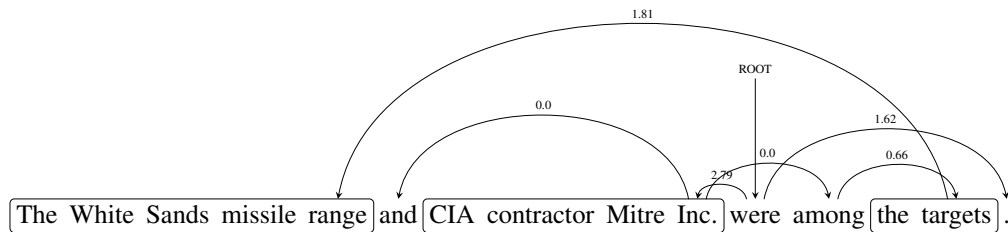


Figure A.5: ILP generated tree with initial weights. Arcs' labels are dependencies' weights

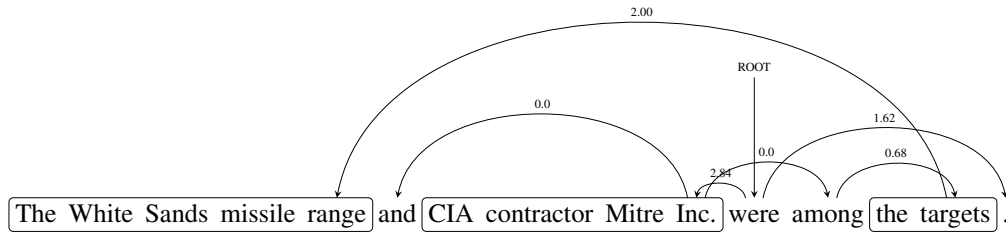


Figure A.6: ILP generated tree with initial weights. Arcs' labels are dependencies' weights

	root	range	and	Inc.	were	among	targets	.
root	0	0	0	0	0	0	0	0
range	1.458	0	0	0	0	0	0	0
and	0.601	1.313	0	0	0	0	0	0
Inc.	1.155	1.806	0.964	0	0	0	0	0
were	1.017	1.736	0.678	2.847	0	0	0	0
among	0.067	0.723	-0.288	1.744	-0.054	0	0	0
targets	1.215	2.008	0.881	2.900	1.100	1.035	0	0
.	0	0.614	-0.356	1.634	-0.128	-0.157	0.097	0
root	0	0.839	0.356	1.507	0.487	0.157	0.336	0
range	0	0	1.436	2.583	1.194	1.223	1.367	1.072
and	0	0	0	3.162	1.624	1.483	1.565	1.326
Inc.	0	0	0	0	1.126	1.178	1.302	0.998
were	0	0	0	0	0	1.316	1.369	1.622
among	0	0	0	0	0	0	0.689	0.254
targets	0	0	0	0	0	0	0	0.844
.	0	0	0	0	0	0	0	0

Table A.6: Objective function (weight matrix) for sentence (6.1) in the second iteration of training with F-WO features.

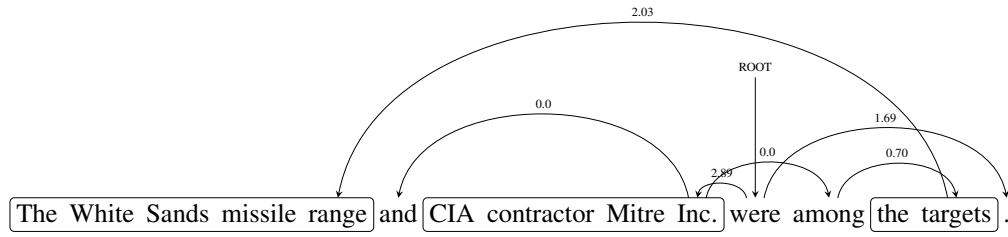


Figure A.7: ILP generated tree with initial weights. Arcs' labels are dependencies' weights.

	root	range	and	Inc.	were	among	targets	.
root	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
range	1.458	0.000	0.000	0.000	0.000	0.000	0.000	0.000
and	0.601	1.319	0.000	0.000	0.000	0.000	0.000	0.000
Inc.	1.180	1.837	0.981	0.000	0.000	0.000	0.000	0.000
were	1.043	1.768	0.704	2.891	0.000	0.000	0.000	0.000
among	0.073	0.735	-0.282	1.769	-0.048	0.000	0.000	0.000
targets	1.229	2.035	0.896	2.933	1.114	1.036	0.000	0.000
.	0.000	0.620	-0.356	1.652	-0.128	-0.160	0.083	0.000
root	0.000	0.842	0.356	1.536	0.487	0.160	0.350	0.000
range	0.000	0.000	1.444	2.619	1.202	1.234	1.375	1.080
and	0.000	0.000	0.000	3.191	1.624	1.486	1.578	1.326
Inc.	0.000	0.000	0.000	0.000	1.161	1.215	1.351	1.033
were	0.000	0.000	0.000	0.000	0.000	1.353	1.408	1.700
among	0.000	0.000	0.000	0.000	0.000	0.000	0.706	0.257
targets	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.838
.	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table A.7: Objective function (weight matrix) for sentence 6.1 in the third iteration of training with F-WO features.

	root	range	and	Inc.	were	among	targets	.
root	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
range	1.491	0.000	0.000	0.000	0.000	0.000	0.000	0.000
and	0.601	1.319	0.000	0.000	0.000	0.000	0.000	0.000
Inc.	1.194	1.850	0.994	0.000	0.000	0.000	0.000	0.000
were	1.043	1.768	0.704	2.918	0.000	0.000	0.000	0.000
among	0.073	0.735	-0.282	1.796	-0.048	0.000	0.000	0.000
targets	1.229	2.035	0.896	2.960	1.114	1.036	0.000	0.000
.	0.000	0.620	-0.356	1.679	-0.128	-0.160	0.083	0.000
root	0.000	0.842	0.356	1.569	0.487	0.160	0.350	0.000
range	0.000	0.000	1.471	2.686	1.229	1.260	1.402	1.106
and	0.000	0.000	0.000	3.225	1.624	1.486	1.578	1.326
Inc.	0.000	0.000	0.000	0.000	1.188	1.242	1.378	1.060
were	0.000	0.000	0.000	0.000	0.000	1.353	1.408	1.700
among	0.000	0.000	0.000	0.000	0.000	0.000	0.706	0.257
targets	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.838
.	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table A.8: Objective function (weight matrix) for sentence (6.1) in the fourth iteration of training with F-WO features.

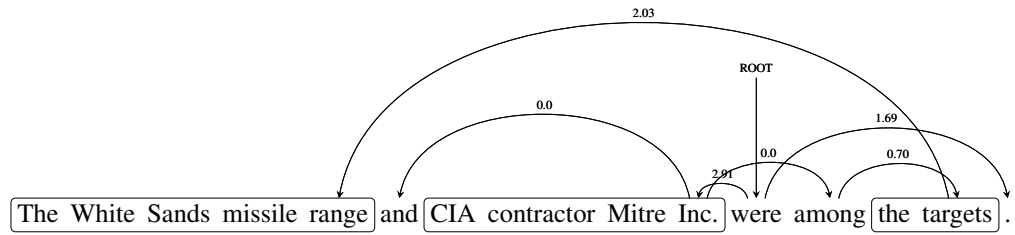


Figure A.8: ILP generated tree with initial weights. Arcs' labels are dependencies' weights

Bibliography

- Tej Anand and Gary Kahn. Making Sense of Gigabytes: A System for Knowledge-Based Market Analysis. In *Proceedings of the fourth conference on Innovative applications of artificial intelligence*, IAAI'92, pages 57—69, San Jose, California, 1992. AAAI Press. 18
- Elisabeth André, Kim Binsted, Kumiko Tanaka-Ishii, Sean Luke, Gerd Herzog, and Thomas Rist. Three RoboCup Simulation League Commentator Systems. *AI Magazine*, 21(1):57–66, 2000. URL citeseer.ist.psu.edu/262208.html. 18
- I Wayan Arka, Avery Andrews, Mary Dalrymple, Meladel Mistica, and Jane Simpson. A linguistic and computational morphosyntactic analysis for the applicative -i in Indonesian. In *Proceedings of LFG 2009*, pages 85–105, Cambridge, UK, 2009. 51
- Armin Bauer, Niki Hoedoro, and Adela Schneider. Rule-based approach to text generation in natural language - automated text markup language (ATML3). In *Proceedings of the RuleML 2015 Challenge, the Special Track on Rule-based Recommender Systems for the Web of Data, the Special Industry Track and the RuleML 2015 Doctoral Consortium hosted by the 9th International Web Rule Symposium (RuleML 2015), Berlin, Germany, August 2-5, 2015.*, 2015. URL <http://ceur-ws.org/Vol-1417/paper20.pdf>. 4
- Anja Belz. Statistical Generation: Three Methods Compared and Evaluated. In *Proceedings of the 10th European Workshop on Natural Language Generation (ENLG 2005)*, pages 15–23, Edinburgh, UK, 2005. 74
- Anja Belz, Mike White, Josef van Genabith, Deirdre Hogan, and Amanda Stent. Finding common ground: Towards a surface realisation shared task. In *Proceedings of the 6th International Natural Language Generation Conference, INLG '10*, pages 268–272, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL <http://dl.acm.org/citation.cfm?id=1873738.1873780>. 35, 36
- Anja Belz, Michael White, Dominic Espinosa, Eric Kow, Deirdre Hogan, and Amanda Stent. The first surface realisation shared task: Overview and evaluation

- results. In *Proceedings of the 13th European workshop on natural language generation*, pages 217–226. Association for Computational Linguistics, 2011. [vii](#), [36](#), [158](#)
- Anja Belz, Bernd Bohnet, Simon Mille, Leo Wanner, and Michael White. The surface realisation task: Recent developments and future plans. In *Proceedings of the Seventh International Natural Language Generation Conference, INLG '12*, pages 136–140, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics. URL <http://dl.acm.org/citation.cfm?id=2392712.2392743>. [36](#)
- Emily Bender. Radical Non-Configurationality without Shuffle Operators: An Analysis of Wambaya. In *Proceedings of the Fifteenth Annual Conference on Head-Driven Phrase Structure Grammar (HPSG08)*, 2008. [51](#)
- Taylor Berg-Kirkpatrick and Dan Klein. Phylogenetic Grammar Induction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. [54](#)
- Dimitri P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, 1982. [127](#)
- Dimitri Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997. [xv](#), [110](#), [111](#)
- Rens Bod. Is the End of Supervised Parsing in Sight? In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 400–407, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/P07-1051>. [54](#)
- Bernd Bohnet. Efficient Parsing of Syntactic and Semantic Dependency Structures. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task, CoNLL '09*, pages 67–72, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. ISBN 978-1-932432-29-9. URL <http://dl.acm.org/citation.cfm?id=1596409.1596421>. [138](#)
- Nadjet Bouayad-Agha, Gerard Casamayor, Leo Wanner, and Chris Mellish. Overview of the First Content Selection Challenge from Open Semantic Web Data. In *Proceedings of the 14th European Workshop on Natural Language Generation*, pages 98–102, Sofia, Bulgaria, 2013. [4](#)
- Sally C. Brailsforda, Chris N. Pottsb, and Barbara M. Smithc. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, 119(3):557–581, December 1999. ISSN 0891-2017. [127](#)
- Joan Bresnan. *Lexical-Functional Syntax*. Blackwell Textbooks in Linguistics.

- Wiley, 2000. ISBN 9780631209737. URL <https://books.google.com.au/books?id=uw62QgAACAAJ>. 24
- Miriam Butt, Helge Dyvik, Tracy Holloway King, Hiroshi Masuichi, and Christian Rohrer. The Parallel Grammar Project. In *Proceedings of COLING-2002 Workshop on Grammar Engineering and Evaluation*, pages 1–7, 2002. 4, 25, 56
- Aoife Cahill, Martin Forst, and Christian Rohrer. Designing Features for Parse Disambiguation and Realisation Ranking. In *Proceedings of the Twelfth International Lexical Functional Grammar Conference*, pages 128–147. CSLI Publications, 2007a. URL <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.162.2643&rep=rep1&type=pdf#page=128>. vii, 27, 28, 95, 99, 103
- Aoife Cahill, Martin Forst, and Christian Rohrer. Stochastic Realisation Ranking for a Free Word Order Language. In *ENLG '07: Proceedings of the Eleventh European Workshop on Natural Language Generation*, pages 17–24, Morristown, NJ, 2007b. Association for Computational Linguistics. vii, 8, 10, 13, 27, 32, 33, 34, 50, 56, 57, 74, 77, 78, 79, 80, 91, 178
- Lynne Cahill, Christy Doran, Roger Evans, Chris Mellish, Daniel Paiva, Mike Reape, Donia Scott, and Neil Tipper. In Search of a Reference Architecture for NLG Systems. Technical report, University of Brighton, 1999. URL <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.7159>. 17
- Songsak Channarukul. YAG: A Template-Based Generator for Real-Time Systems. Master thesis, Department of Electrical Engineering and Computer Science, University of Wisconsin-Milwaukee, December 1999. 4
- Eugene Charniak. Immediate-Head Parsing for Language Models. In *Proceedings of 39th Annual Meeting of the Association for Computational Linguistics (ACL'01)*, pages 124–131, Toulouse, France, 2001. xiv, 52
- Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 173–180, Ann Arbor, Michigan, 2005. 57, 78
- Eugene Charniak, Kevin Knight, and Kenji Yamada. Syntax-based Language Models for Machine Translation. In *Proceedings of MT Summit IX*, 2003. 53
- Colin Cherry and Chris Quirk. Discriminative, Syntactic Language Modeling through Latent SVMs. In *Proceedings of the Eighth Conference of the Association for Machine Translation in the Americas (AMTA'08)*, 2008. 53
- Stephen Clark and James R. Curran. Wide-Coverage Efficient Statisti-

- cal Parsing with CCG and Log-Linear Models. *Computational Linguistics*, 33(4):493–552, 2007. ISSN 08912017. doi: 10.1162/coli.2007.33.4.493. URL <http://www.mitpressjournals.org/doi/abs/10.1162/coli.2007.33.4.493>. 35
- Michael John Collins. A New Statistical Parser Based on Bigram Lexical Dependencies. In *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, ACL '96, pages 184–191, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics. doi: 10.3115/981863.981888. URL <http://dx.doi.org/10.3115/981863.981888>. 40, 41
- Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001. ISBN 0070131511. 131
- Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online Passive-Aggressive Algorithms. *J. Mach. Learn. Res.*, 7: 551–585, December 2006. ISSN 1532-4435. URL <http://dl.acm.org/citation.cfm?id=1248547.1248566>. 38, 137
- Mary Dalrymple. *Lexical-Functional Grammar (Syntax and Semantics, Volume 34) (Syntax and Semantics) (Syntax and Semantics)*. Emerald Group Publishing Limited, 2001. ISBN 0126135347. 24
- Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. Generating Typed Dependency Parses from Phrase Structure Parses. In *Proceedings of LREC*, volume 6, pages 449–454. Genoa Italy, 2006. 126
- Koenraad De Smedt, Helmut Horacek, and Michael Zock. Architectures for Natural Language Generation: Problems and Perspectives. In *Trends In Natural Language Generation: An Artificial Intelligence Perspective*, Lecture Notes in Computer Science, pages 17–46. Springer-Verlag, 1996. URL <http://www.springerlink.com/content/71g3256h3431j37n/fulltext.pdf>. 1
- Simon Dennis. *Introducing Word Order within the LSA Framework*, chapter 22. Psychology Press, May 2013. 105
- Stefanie Dipper. *Implementing and Documenting Large Scale Grammars: German LFG*. Arbeitspapiere des Instituts für Maschinelle Sprachverarbeitung. Inst. für Maschinelle Sprachverarbeitung, 2003. URL <https://books.google.com.au/books?id=JP4EtwAACAAJ>. 25
- Mark Dras. Squibs: Evaluating human pairwise preference judgments. *Computational Linguistics*, 41(2):309–317, 2015. doi: 10.1162/COLI_a_00222. URL <http://www.aclweb.org/anthology/J15-2005>. 57
- Mark Dras, François Lareau, Benjamin Börschinger, Robert Dale, Yasaman Mo-

- tazedi, Owen Rambow, Myfany Turpin, and Morgan Ulinski. Complex Predicates in Arrernte. In *Proceedings of LFG 2012*, Bali, Indonesia, 2012. 51
- Jason M. Eisner. Three New Probabilistic Models for Dependency Parsing: An Exploration. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 1*, COLING '96, pages 340–345, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics. doi: 10.3115/992628.992688. URL <http://dx.doi.org/10.3115/992628.992688>. 141
- Roger Evans, Paul Piwek, and Lynne Cahill. What is NLG. In *Proceedings of the second International Conference on Natural Language Generation*, pages 144–151, 2002. 15, 16
- Martin Forst. "Disambiguation for a Linguistically Precise German Parser". PhD thesis, University of Stuttgart, 2007. 27, 31
- Eli Goldberg, Norbert Driedger, and Richard I Kittredge. Using Natural-Language Processing to Produce Weather Forecasts. *Ieee Expert Intelligent Systems And Their Applications*, 9(2):45–53, 1994. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=294135. 1, 17
- Yoav Goldberg. A Primer on Neural Network Models for Natural Language Processing. *CoRR*, abs/1510.00726, 2015. URL <http://arxiv.org/abs/1510.00726>. xiv, 46, 47
- Yoav Goldberg and Jon Orwant. A Dataset of Syntactic-Ngrams over Time from a Very Large Corpus of English Books. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 241–247, Atlanta, Georgia, USA, June 2013. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/S13-1035>. 123
- Keith Hall. K-best Spanning Tree Parsing. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL'07)*, pages 392–399, Prague, Czech Republic, 2007. Association for Computational Linguistics. 74
- William P. Headden III, Mark Johnson, and David McClosky. Improving Unsupervised Dependency Parsing with Richer Contexts and Smoothing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'09)*, pages 101–109, Boulder, Colorado, 2009. 54
- Hideki Hirakawa. Graph Branch Algorithm: An Optimum Tree Search Method for Scored Dependency Graph with Arc Co-occurrence Constraints. In *Proceedings of the COLING/ACL on Main Conference Poster Sessions*, COLING-ACL '06, pages 361–368, Stroudsburg, PA, USA, 2006. Association for Compu-

- tational Linguistics. URL <http://dl.acm.org/citation.cfm?id=1273073.1273120>. 40
- Julia Hockenmaier and Mark Steedman. CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396, 2007. ISSN 08912017. doi: 10.1162/coli.2007.33.3.355. URL <http://www.mitpressjournals.org/doi/abs/10.1162/coli.2007.33.3.355>. 35, 43
- Richard Johansson and Pierre Nugues. Extended Constituent-to-dependency Conversion for English. In *Proceedings of NODALIDA 2007*, pages 105–112, Tartu, Estonia, May 25–26 2007. 156
- Bryant A. Julstrom. Comparing Decoding Algorithms in a Weight-coded GA for TSP. In *Proceedings of the 1998 ACM Symposium on Applied Computing, SAC '98*, pages 313–317, New York, NY, USA, 1998. ACM. ISBN 0-89791-969-6. doi: 10.1145/330560.330830. URL <http://doi.acm.org/10.1145/330560.330830>. 130
- Sylvain Kahane and François Lareau. *Word Ordering as a Graph Rewriting Process*, pages 216–239. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016. ISBN 978-3-662-53042-9. doi: 10.1007/978-3-662-53042-9_13. URL http://dx.doi.org/10.1007/978-3-662-53042-9_13. 45
- Andrej Karpathy and Fei-Fei Li. Deep visual-semantic alignments for generating image descriptions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7–12, 2015*, pages 3128–3137, 2015. doi: 10.1109/CVPR.2015.7298932. URL <http://dx.doi.org/10.1109/CVPR.2015.7298932>. 48
- Sami Khuri and Thomas Bäck. An Evolutionary Heuristic for the Minimum Vertex Cover Problem. In *KI-94 Workshops (Extended Abstracts)*, pages 86–90, 1994. 130
- Dan Klein and Christopher Manning. Corpus-Based Induction of Syntactic Structure: Models of Dependency and Constituency. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04)*, pages 478–485, Barcelona, Spain, 2004. 53, 55
- Dan Klein and Christopher D. Manning. Accurate Unlexicalized Parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan, 2003. 57
- Adrienne Lafrance. A Computer Tried (and Failed) to Write This Article. *The Atlantic*, 2016. URL <http://www.theatlantic.com/technology/archive/2016/06/story-by-a-human/485984/>. 3
- I. Langkilde and K. Knight. Generation that Exploits Corpus-based Statistical

- Knowledge. In *Proceedings of the ACL/COLING-98*, Montreal, Quebec, 1998. 11
- Irene Langkilde and Kevin Knight. In *Proceedings of the Ninth International Workshop on Natural Language Generation*, Ontario, Canada, *interhash* = 29657ee9b7bc20564aaa6c50a375f10d, *intrahash* = 3028917190ac3e66a790ba7874b18a51, *keywords* = 2000 book nlp, *pages* = 248–255, *timestamp* = 2008-03-17T11:27:54.000+0100, *title* = "The Practical Value of N-Grams in Generation", *year* = 1998. 8, 29, 105
- Irene Langkilde-Geary. Forest-Based Statistical Sentence Generation. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 170–177. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2000. URL <http://portal.acm.org/citation.cfm?id=974328&dl=GUIDE, #url.dl>. 19
- Irene Langkilde-Geary. An Empirical Verification of Coverage and Correctness for a General-Purpose Sentence Generator. In *Proceedings of the 12th International Natural Language Generation Workshop*, pages 17–24. Citeseer, 2002. URL <http://www.cs.rutgers.edu/~mdstone/inlg02/112.pdf>. 20
- Benoit Lavoie and Owen Rambow. A Fast and Portable Realizer for Text Generation Systems. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, number 1, pages 265–268. Association for Computational Linguistics, ACL, 1997. doi: 10.1007/s10689-010-9322-0. URL citeseer.ist.psu.edu/lavoie97fast.html. 17
- Christopher D. Manning. Computational Linguistics and Deep Learning. *Comput. Linguist.*, 41(4):701–707, December 2015. ISSN 0891-2017. doi: 10.1162/COLI_a_00239. URL http://dx.doi.org/10.1162/COLI_a_00239. 48
- Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The Penn Treebank: Annotating Predicate Argument Structure. In *Proceedings of the Workshop on Human Language Technology, HLT '94*, pages 114–119, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics. ISBN 1-55860-357-3. doi: 10.3115/1075812.1075835. URL <http://dx.doi.org/10.3115/1075812.1075835>. 156
- André F. T. Martins, Noah A. Smith, and Eric P. Xing. Concise Integer Linear Programming Formulations for Dependency Parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL '09, pages 342–350, Stroudsburg, PA, USA, 2009. Association

- for Computational Linguistics. ISBN 978-1-932432-45-9. URL <http://dl.acm.org/citation.cfm?id=1687878.1687928>. 108, 115
- M. Mauldin. Semantic rule Based Text Generation. In *COLING-84*, Stanford, 1984. 4
- Ryan McDonald. *Discriminative Learning and Spanning Tree Algorithms for Dependency Parsing*. PhD thesis, University of Pennsylvania, May 2006. ix, 140
- Ryan McDonald and Giorgio Satta. On the Complexity of Non-projective Data-driven Dependency Parsing. In *Proceedings of the 10th International Conference on Parsing Technologies, IWPT '07*, pages 121–132, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics. ISBN 978-1-932432-90-9. URL <http://dl.acm.org/citation.cfm?id=1621410.1621426>. 40, 43
- Ryan McDonald, Koby Crammer, and Fernando Pereira. Online Large-margin Training of Dependency Parsers. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, ACL '05*, pages 91–98, Stroudsburg, PA, USA, 2005a. Association for Computational Linguistics. doi: 10.3115/1219840.1219852. URL <http://dx.doi.org/10.3115/1219840.1219852>. 38, 135, 137, 139, 161
- Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-projective Dependency Parsing Using Spanning Tree Algorithms. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing, HLT '05*, pages 523–530, Stroudsburg, PA, USA, 2005b. Association for Computational Linguistics. doi: 10.3115/1220575.1220641. URL <http://dx.doi.org/10.3115/1220575.1220641>. xiii, 11, 12, 38, 39, 40, 108, 113, 115, 139, 141, 143, 161
- Ryan McDonald, Slav Petrov, and Keith Hall. Multi-source Transfer of Delexicalized Dependency Parsers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP '11*, pages 62–72, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 978-1-937284-11-4. URL <http://dl.acm.org/citation.cfm?id=2145432.2145440>. 74
- Ryan T McDonald and Fernando CN Pereira. Online Learning of Approximate Dependency Parsing Algorithms. In *EACL*, 2006. URL <https://www.aclweb.org/anthology/E/E06/E06-1011.pdf>. 40
- Kathleen McKeown, Karen Kukich, and James Shaw. Practical Issues in Automatic Documentation Generation. In *Proceedings of the fourth conference on Applied natural language processing*, page 7. Association for Computational Linguistics,

1994. doi: 10.3115/974358.974361. URL <http://portal.acm.org/citation.cfm?doid=974358.974361>. 17
- Chris Mellish, Donia Scott, Lynne Cahill, Daniel Paiva, Roger Evans, and Mike Reape. A Reference Architecture for Natural Language Generation Systems. *Natural Language Engineering*, 12(1):1–34, 2006. URL <http://oro.open.ac.uk/2326/>. 17
- Zbigniew Michalewicz. A survey of Constraint Handling Techniques in Evolutionary Computation Methods. In *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pages 135–155. MIT Press, 1995. 127
- Yasaman Motazedi, Mark Dras, and François Lareau. Is Bad Structure Better Than No Structure?: Unsupervised Parsing for Realisation Ranking. In *COLING 2012, 24th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, 8-15 December 2012, Mumbai, India*, pages 1811–1830, 2012. URL <http://aclweb.org/anthology/C/C12/C12-1111.pdf>. 51
- Stefan Müller. Persian Complex Predicates and the Limits of Inheritance-Based Analyses. *Journal of Linguistics*, 46(3):601–655, 2010. 51
- Andrew Mutton, Mark Dras, Stephen Wan, and Robert Dale. GLEU: Automatic Evaluation of Sentence-Level Fluency. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL'07)*, pages 344–351, Prague, Czech Republic, 2007. 53
- Tahira Naseem. *Disambiguation for a Linguistically Precise German Parser*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2014. 74
- Tahira Naseem, Harr Chen, Regina Barzilay, and Mark Johnson. Using Universal Linguistic Knowledge to Guide Grammar Induction. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP'10)*, pages 1234–1244, Cambridge, MA, 2010. vii, 54, 55, 58
- George Nemhauser and Laurence Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1988. ISBN 9780471828198. URL <https://books.google.com.au/books?id=uG4PAQAAMAAJ>. 109
- Frank Neumann and Ingo Wegener. Minimum spanning trees made easier via multi-objective optimization. *Natural Computing*, 5(3):305–319, September 2007. ISSN 1567-7818. 130
- Anton Nijholt, Rieks Den Akker, and Franciska De Jong. Language Interpretation and Generation for Football Commentary. *ACTAS-1: VIII Symposio Social*, 2003. 18

- Jon Oberlander and Chris Brew. Stochastic Text Generation. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, 358(1769): 1373–1387, 2000. 5
- Stephan Oepen, Kristina Toutanova, Stuart Shieber, Christopher Manning, Dan Flickinger, and Thorsten Brants. The LinGO Redwoods Treebank Motivation and Preliminary Applications. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 2, COLING '02*, pages 1–5, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1071884.1071909. URL <http://dx.doi.org/10.3115/1071884.1071909>. 23
- Charles C. Palmer and Aron Kershenbaum. Representing Trees in Genetic Algorithms. In *the First IEEE Conference on Evolutionary Computation*, pages 379–384. IEEE Press., October 5-7 1994. 130
- Richard Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL'02)*, pages 311–318, Philadelphia, 2002. 33
- Matt Post and Daniel Gildea. Parsers as Language Models for Statistical Machine Translation. In *Eighth Conference of the Association for Machine Translation in the Americas (AMTA 2008)*, Honolulu, HI, 2008. 53
- R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957. ISSN 1538-7305. doi: 10.1002/j.1538-7305.1957.tb01515.x. URL <http://dx.doi.org/10.1002/j.1538-7305.1957.tb01515.x>. 131
- Günther R. Raidl. A Weight-coded Genetic Algorithm for the Multiple Container Packing Problem. In *Proceedings of the 1999 ACM Symposium on Applied Computing, SAC '99*, pages 291–296, New York, NY, USA, 1999. ACM. ISBN 1-58113-086-4. doi: 10.1145/298151.298354. URL <http://doi.acm.org/10.1145/298151.298354>. 130
- Günther R. Raidl and Bryant A. Julstrom. A Weighted Coding in a Genetic Algorithm for the Degree-constrained Minimum Spanning Tree Problem. In *Proceedings of the 2000 ACM Symposium on Applied Computing - Volume 1, SAC '00*, pages 440–445, New York, NY, USA, 2000. ACM. ISBN 1-58113-240-9. doi: 10.1145/335603.335888. URL <http://doi.acm.org/10.1145/335603.335888>. 131
- Jayant Rajgopal. Principles and Applications of Operations Research. *Maynard's Industrial Engineering Handbook*, pages 11.27–11.44, 2004. 110
- Rajakrishnan Rajkumar and Michael White. Designing Agreement Features for

- Realization Ranking. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, number August in COLING '10, pages 1032–1040. Association for Computational Linguistics, 2010. URL <http://portal.acm.org/citation.cfm?id=1944566.1944685>. [xiii](#), [34](#)
- Adwait Ratnaparkhi. Trainable Methods for Surface Natural Language Generation. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 194—201, Seattle, Washington, 2000. Morgan Kaufmann Publishers Inc. URL <http://arxiv.org/abs/cs/0006028>. [xiii](#), [20](#), [21](#)
- Ehud Reiter and Robert Dale. Building applied natural language generation systems. *Nat. Lang. Eng.*, 3(1):57–87, March 1997. ISSN 1351-3249. doi: 10.1017/S1351324997001502. URL <http://dx.doi.org/10.1017/S1351324997001502>. [105](#)
- Ehud Reiter and Robert Dale. *Building Natural Language Generation Systems*. Number Cambridge in Studies in Natural Language Processing. Cambridge University Press, 2000. ISBN 0521620368. doi: 10.2277/052102451X. URL <http://assets.cambridge.org/052162/0368/sample/0521620368WSN01.pdf>. [vii](#), [xiii](#), [1](#), [2](#), [15](#), [16](#), [18](#)
- Sebastian Riedel and James Clarke. Incremental Integer Linear Programming for Non-projective Dependency Parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP '06, pages 129–137, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics. ISBN 1-932432-73-6. URL <http://dl.acm.org/citation.cfm?id=1610075.1610095>. [40](#), [108](#), [113](#), [114](#), [115](#), [117](#), [119](#), [125](#)
- Stefan Riezler and Alexandre Vasserman. Gradient Feature Testing and L1 Regularization for Maximum Entropy Parsing. In *Proceedings of EMNLP'04*, Barcelona, Spain., 2004. [vii](#), [27](#), [28](#), [31](#)
- Stefan Riezler, Tracy H King, Ronald M Kaplan, Richard Crouch, John T Maxwell III, and Mark Johnson. Parsing the Wall Street Journal using a Lexical-Functional Grammar and Discriminative Estimation Techniques. In *Proceedings of the 40th Annual Meeting of the ACL*, number July, pages 271–278. Association for Computational Linguistics, 2002. doi: 10.3115/1073083.1073129. URL <http://portal.acm.org/citation.cfm?id=1073129&dl=GUIDE>, . [vii](#), [25](#), [27](#), [28](#), [31](#)
- Christian Rohrer and Martin Forst. Improving Coverage and Parsing Quality of a Large-scale LFG for German. In *LREC-2006*, Genoa, Italy., 2006. [25](#)
- Chakaveh Saedi, Mehrnoush Shamsfard, and Yasaman Motazedi. Automatic Translation between English and Persian Texts. In *In Proceedings of the Third*

- Workshop on Computational Approaches to Arabic Script-based Languages*, 2009. [10](#)
- Kevin Scannell. The Crúbadán Project: Corpus building for under-resourced languages. In C. Fairon, H. Naets, A. Kilgarrieff, and G-M. de Schryver, editors, *Building and Exploring Web Corpora: Proceedings of the 3rd Web as Corpus Workshop*, volume 4 of *Cahiers du Cental*, pages 5–15. Presses universitaires de Louvain, Louvain-la-Neuve, Belgium, 2007. [58](#), [73](#)
- Mehrnoush Shamsfard. Developing FarsNet: A Lexical Ontology for Persian. In *4th Global WordNet Conference, Szeged, Hungary*, 2008. [10](#)
- Alice E. Smith and David W. Coit. Constraint Handling Techniques - Penalty Functions. In Thomas Baeck, David Fogel, and Zbigniew Michalewicz, editors, *Oxford Handbook of Innovation*, chapter C5.2. Institute of Physics Publishing and Oxford University Press, Bristol U.K., 1997. [127](#), [128](#), [129](#)
- Mark Steedman. *The Syntactic Process*, volume 131 of *Language, Speech, and Communication series*. MIT Press, 2000. ISBN 0262194201. URL <http://www.mitpressjournals.org/doi/pdfplus/10.1162/coli.2000.27.1.146>. [35](#)
- A. Stolcke. SRILM – An Extensible Language Modeling Toolkit. In *Proc. Intl. Conf. on Spoken Language Processing*, pages 901–904, Denver, CO, US, 2002. [58](#)
- Radhika Subramanian, Richard P Scheff, John D Quillinan, D Steve Wiper, and Roy E Marsten. Coldstart: Fleet Assignment at Delta Air Lines. *Interfaces*, 24 (1):104–120, 1994. [111](#)
- Sebastian Sulger, Miriam Butt, Tracy King, Paul Meurer, Tibor Laczkó, György Rákosi, Bamba Dione, Helge Dyvik, Victoria Rosñ, Koenraad De Smedt, Agnieszka Patejuk, Ozlem Cetinoglu, I Wayan Arka, and Meladel Mistica. ParGramBank: The ParGram parallel treebank. In *ACL 2013 - 51st Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, volume 1, pages 550–560, 08 2013. [25](#)
- Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011. [48](#)
- Oscar Täckström, Ryan McDonald, and Joakim Nivre. Target Language Adaptation of Discriminative Transfer Parsers. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1061–1071, Atlanta, Georgia, June 2013. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/N13-1126>. [74](#)

- Ben Taskar, Dan Klein, Michael Collins, Daphne Koller, and Christopher Manning. Max-margin Parsing. In *In Proceedings of EMNLP*, 2004. 40
- Mariet Theune, E Klabbers, J Odijk, Jan Roelof de Pijper De Pijper, and Emiel Krahmer. From data to speech: general approach. *Natural Language Engineering*, 7(1):47—86, 2001. URL <http://portal.acm.org/citation.cfm?id=973927.973930>. 17
- Erik Velldal and Stephan Oepen. Maximum entropy models for realization ranking. In *Proceedings of the 10th MT Summit*, pages 109–116, Thailand, 2005. xiii, 19, 23, 29, 35
- Erik Velldal, Stephan Oepen, and Dan Flickinger. Paraphrasing treebanks for stochastic realization ranking. In *Proceedings of TLT Workshop*, pages 149–160, Tübingen, 2004. 23, 33
- Stephen Wan, Mark Dras, Robert Dale, and Cécile Paris. Improving Grammaticality in Statistical Sentence Generation: Introducing a Dependency Spanning Tree Algorithm with an Argument Satisfaction Model. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '09, pages 852–860, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. URL <http://dl.acm.org/citation.cfm?id=1609067.1609162>. xiv, xv, 11, 14, 40, 42, 43, 45, 50, 105, 106, 107, 108, 116, 117, 134, 137, 138, 153, 156, 161, 168, 175, 178, 179
- Zhiguo Wang and Chengqing Zong. Parse Reranking Based on Higher-Order Lexical Dependencies. In *Proceedings of 5th International Joint Conference on Natural Language Processing (IJCNLP'11)*, pages 1251–1259, Chiang Mai, Thailand, 2011. 74
- Tsung-Hsien Wen, Milica Gasic, Dongho Kim, Nikola Mrksic, Pei-hao Su, David Vandyke, and Steve J. Young. Stochastic Language Generation in Dialogue using Recurrent Neural Networks with Convolutional Sentence Reranking. *CoRR*, abs/1508.01755, 2015. URL <http://arxiv.org/abs/1508.01755>. 48
- Yiming Yang and Jan O. Pedersen. A Comparative Study on Feature Selection in Text Categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML'97)*, pages 412–420, 1997. 62
- Daniel Zeman, David Mareček, Martin Popel, Loganathan Ramasamy, Jan Štěpánek, Zdeněk Žabokrtský, and Jan Hajič. HamleDT: To Parse or Not to Parse? In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, 2012. 68, 74
- Yue Zhang. Partial-tree Linearization: Generalized Word Ordering for Text Synthesis. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI'13*, pages 2232–2238. AAAI Press, 2013. ISBN

978-1-57735-633-2. URL <http://dl.acm.org/citation.cfm?id=2540128.2540449>. 5, 11, 12, 45, 105

Yue Zhang and Stephen Clark. A Tale of Two Parsers: Investigating and Combining Graph-based and Transition-based Dependency Parsing Using Beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 562–571, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics. URL <http://dl.acm.org/citation.cfm?id=1613715.1613784>. 123

Yue Zhang and Stephen Clark. Syntax-based grammaticality improvement using ccg and guided search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1147–1157, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 978-1-937284-11-4. URL <http://dl.acm.org/citation.cfm?id=2145432.2145554>. 11, 43, 45, 108, 116, 175

Yue Zhang and Joakim Nivre. Transition-based Dependency Parsing with Rich Non-local Features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, pages 188–193, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 978-1-932432-88-6. URL <http://dl.acm.org/citation.cfm?id=2002736.2002777>. 123

Yue Zhang, Graeme Blackwood, and Stephen Clark. Syntax-based Word Ordering Incorporating a Large-scale Language Model. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '12, pages 736–746, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics. ISBN 978-1-937284-19-0. URL <http://dl.acm.org/citation.cfm?id=2380816.2380906>. 45, 116, 175