EXACT SAMPLING AND OPTIMISATION IN STATISTICAL MACHINE TRANSLATION

Wilker Ferreira Aziz

A thesis submitted in partial fulfilment of the requirements of the University of Wolverhampton for the degree of Doctor of Philosophy

2014

Signature: . . . .

Date: . . . .

Abstract

In Statistical Machine Translation (SMT), inference needs to be performed over a high-complexity discrete distribution defined by the intersection between a translation hypergraph and a target language model. This distribution is too complex to be represented exactly and one typically resorts to approximation techniques either to perform *optimisation* – the task of searching for the optimum translation – or *sampling* – the task of finding a subset of translations that is statistically representative of the goal distribution. Beam-search is an example of an approximate optimisation technique, where maximisation is performed over a heuristically pruned representation of the goal distribution.

For inference tasks other than optimisation, rather than finding a single optimum, one is really interested in obtaining a set of probabilistic *samples* from the distribution. This is the case in training where one wishes to obtain unbiased estimates of expectations in order to fit the parameters of a model. Samples are also necessary in consensus decoding where one chooses from a sample of likely translations the one that minimises a loss function. Due to the additional computational challenges posed by sampling, n-best lists, a by-product of optimisation, are typically used as a biased approximation to true probabilistic samples. A more direct procedure is to attempt to directly draw samples from the underlying distribution rather than rely on n-best list approximations.

Markov Chain Monte Carlo (MCMC) methods, such as Gibbs sampling, offer a way to overcome the tractability issues in sampling, however their convergence properties are hard to assess. That is, it is difficult to know when, if ever, an MCMC sampler is producing samples that are compatible

with the goal distribution. Rejection sampling, a Monte Carlo (MC) method, is more fundamental and natural, it offers strong guarantees, such as unbiased samples, but is typically hard to design for distributions of the kind addressed in SMT, rendering an intractable method.

A recent technique that stresses a unified view between the two types of inference tasks discussed here — optimisation and sampling — is the OS* approach. OS* can be seen as a cross between Adaptive Rejection Sampling (an MC method) and A* optimisation. In this view the intractable goal distribution is upperbounded by a simpler (thus tractable) proxy distribution, which is then incrementally refined to be closer to the goal until the maximum is found, or until the sampling performance exceeds a certain level.

This thesis introduces an approach to exact optimisation and exact sampling in SMT by addressing the tractability issues associated with the intersection between the translation hypergraph and the language model. The two forms of inference are handled in a unified framework based on the OS* approach. In short, an intractable *goal distribution*, over which one wishes to perform inference, is *upperbounded* by tractable *proposal distributions*. A proposal represents a *relaxed* version of the complete space of weighted translation derivations, where relaxation happens with respect to the incorporation of the language model. These proposals give an optimistic view on the true model and allow for easier and faster search using standard dynamic programming techniques. In the OS* approach, such proposals are used to perform a form of adaptive rejection sampling. In rejection sampling, samples are drawn from a proposal distribution and accepted or rejected as a function of the mismatch between the proposal and the goal. The technique is adaptive in that rejected samples are used to motivate a refinement of the upperbound proposal that brings it closer to the goal, improving the rate of acceptance. Optimisation can be connected to an extreme form of sampling, thus the framework introduced here suits both exact optimisation and exact

sampling. *Exact optimisation* means that the global maximum is found with a certificate of optimality. *Exact sampling* means that unbiased samples are independently drawn from the goal distribution.

We show that by using this approach exact inference is feasible using only a fraction of the time and space that would be required by a full intersection, without recourse to pruning techniques that only provide approximate solutions. We also show that the vast majority of the entries (n-grams) in a language model can be summarised by shorter and optimistic entries. This means that the computational complexity of our approach is less sensitive to the order of the language model distribution than a full intersection would be. Particularly in the case of sampling, we show that it is possible to draw exact samples compatible with distributions which incorporate a high-order language model component from proxy distributions that are much simpler. In this thesis, exact inference is performed in the context of both hierarchical and phrase-based models of translation, the latter characterising a problem that is NP-complete in nature.

Acknowledgements

Before you read this text you should probably know that it would not have been possible without the input and support of a few truly remarkable people — some of which you might know, some of which you will never meet, but all of them immensely important to me. Before I tell you who they are, you should know that any mistakes left in this text is my fault alone.

I am going to start by thanking my advisors. A huge thank you to Lucia Specia, who advises me since before the PhD, who taught me what it means to be a researcher and with whom I learnt so much about machine translation. I first met Lucia when I knew very little about MT and research in general. Yet, probably thanks to Thiago Pardo's and Ivandré Paraboni's kind words about my undergrad work, she trusted me with an internship at Xerox (where I also first met Marc). Since those days Lucia has been an incessant source of ideas, valuable criticism and support. I am fortunate for our collaboration will continue (she hired me as a postdoc at the University of Sheffield — yay!). I would like to thank Prof. Ruslan Mitkov for all his support and trust throughout the 3 years in which I worked with my colleagues at the University of Wolverhampton. Last but far from least, I would like to thank Marc Dymetman. Even though Marc joined the supervisory team late in this journey, his ideas and input changed the course of my studies to what you will see here. Marc introduced me to a much more formal world where I found most of what makes me passionate about statistical machine translation. Marc's eye for detail always pushed me to learn more about everything I ever had to do. I owe him a huge thank you, so big that the finest bottle of French wine I can afford won't do the job. I also thank Sriram Venkatapathy who, together with Marc, generously invested his time in several inspirational brainstorming sessions.

I would like to thank my examiners Gloria Corpas and Khalil Sima'an who gave me the opportunity to discuss my work at a challenging and exciting level. Particularly, Khalil showed an exciting interest for my research and gave me lots of great ideas. Had I had the chance to brainstorm with him one year earlier, this thesis would have been a lot longer!

The following are the people you most likely don't know and probably won't ever meet, but who represented and unmeasurable amount of support. My partner in all causes, at all times and who surrounded me with love regardless of my mood, Marialaura Giova, who also shared with me her family when mine was far (thank you Enzo, Tonia and Antonio). My friend and brother with whom I shared most of my 3 years in England and with whom I hope to share many more, Miguel Rios. Friends with whom I spent my funniest times: Ismail and Albert (first friends I made in Europe), José Guilherme and Iacer (these guys made my life in Wolverhampton so much better), Serena (biondíssima), Iustin (who always made it sound like my work was so exciting), Teo and Carol (who together with Marialaura kept me sane in Sheffield at the time I was writing this text). All my colleagues in Wolverhampton who every now and then would join me in making cold England a warmer place to live. And finally, my friends and brothers Mateus (Fagá) and Luiz (Cazé) who pushed me to academia, always supportive of me despite the distance, and who never let our friendship change, not a tiny bit, thank you guys!

To conclude, I would like to thank my parents (Edna and Marcos) and my sisters (Thabata and Bianca) for never doubting me, for always supporting me and for their unconditional love. Special thanks to my mum, who made it from Brazil to England to be there and hug me after my PhD viva.

Thank you all!

Contents

List of Tables

List of Figures

Acronyms

AGTSP	Asymmetric Generalised TSP
BLEU	Bilingual Evaluation Understudy
CFG	Context-Free Grammar
DP	Dynamic Programming
FSA	Finite-State Automaton
FST	Finite-State Transducer
GNF	Greibach Normal Form
ILP	Integer Linear Program
ITG	Inversion Transduction Grammar
HMM	Hidden Markov Model
LM	Language Model
LR	Lagrangian Relaxation
MBR	Minimum Bayes Risk
MC	Monte Carlo
MCMC	Markov Chain Monte Carlo
MERT	Minimum Error Rate Training
MT	Machine Translation

MTG	Multitext Grammar
NLP	Natural Language Processing
PCFG	Probabilistic Weighted Context-Free Grammar
PDA	Push-Down Automaton
wSCFG	Weighted Synchronous Context-Free Grammar
SCFG	Synchronous Context-Free Grammar
SMT	Statistical Machine Translation
STD	Syntax-Directed Transducer
TAG	Tree-Adjoining Grammar
TSG	Tree-Substitution Grammar
TSP	Travelling Salesman Problem
wCFG	Weighted Context-Free Grammar
wFSA	Weighted Finite-State Automaton
wFST	Weighted Finite-State Transducer
wSCFG	Weighted Synchronous Context-Free Grammar
WMT	Workshop on Statistical Machine Translation

Symbols

x	input symbol
$\mathbf{x}$	input vector — typically the input sentence
y	output symbol
$\mathbf{y}$	output vector — typically the target yield of a derivation
$\mathbf{d}$	derivation
$\mathcal{D}$	set of derivations
e	edge (or transition)
I	input length
J	output length
f	objective function
Λ	feature weights
λ	feature weight
H	feature function
$\mathbf{H}$	vector of feature functions
h	local feature function
$\mathbf{h}$	vector of local feature functions
Σ	input vocabulary
Δ	output vocabulary
p	goal distribution
q	proposal distribution
G	translation forest or lattice
A	automaton
$\langle x_i^{i'}, y_j^{j'} \rangle$	a biphrase
n	typically the order of the language model

l	typically number of steps in a derivation
ϕ	typically the weighted translation features
ψ	typically the weighted language model features
δ	typically the weighted distortion cost

Chapter 1

Introduction

Statistical Machine Translation (SMT) emerged as the predominant approach to Machine Translation (MT). It frames the task as a machine learning problem and it relies on statistics to chose amongst a huge set of possible answers the one which best fits what was previously seen as good translations.

In SMT a model of *translational equivalences* explains all known correspondences between a source language and a target language. These correspondences are the building blocks of translation, for example, they may consist of pairs of phrases (Koehn et al., 2003) or pairs of rewrite rules (Chiang, 2005), which are automatically learnt from parallel corpora, i.e. bilingual collections of text. Typically this task relies on word-alignments, word-to-word correspondences in the bilingual corpora, which are also automatically inferred from unlabelled data, for instance via expectation maximisation (Brown et al., 1993; Och and Ney, 2003).[1] A *parameterisation* of the model assigns a score to each possible way of translating an input text in terms of its building blocks, a sequence of steps called a *derivation*. This parameterisation is typically expressed as a linear combination of feature functions that decompose additively over the sequence of steps in the derivation. These feature functions are assumed to assess independent aspects of the correspondences between input and output as established by a derivation. The linear combination utilises a set of scaling *parameters* that capture the relative importance of each feature function, these allow one to optimise the

[1]Other formulations enable the learning of pairs of phrases or pairs of rewrite rules to be done directly from unannotated data (Marcu and Wong, 2002; Blunsom et al., 2009).

model's accuracy using machine learning algorithms, a task called *parameter estimation* (Och, 2003; Kumar and Byrne, 2004). Finally, a *decision rule* selects out of the space of weighted derivations the one that satisfies a certain criterion, such as optimality under the parameterisation.[2]

The most popular decision rule in SMT is the one that searches for the derivation with maximum score under the parameterisation of the model, also known as *decoding*. This task involves searching through a large combinatorial space of derivations representing all possible ways of covering the input yielding a translation. The characteristics of this space depends on the formalism used to represent translational equivalences and on the choice of parameterisation. Translational equivalences are typically represented with finite-state transducers or synchronous context-free grammars. Phrase-based translation (Koehn et al., 2003) is an instance of the former, and hierarchical phrase-based translation (Chiang, 2005) is an instance of the latter.

In a more abstract way, decoding involves two steps. First, the model of translational equivalences is applied to the input, which results in a translation *hypergraph*. The derivations in this hypergraph can be weighted to account for some form of *local parameterisation* (e.g. phrase/rule translation probabilities). Second, a Language Model (LM) is used to rescore the derivations in the hypergraph with respect to the translation strings they define. In language modelling, a sequence of target words is scored under the assumption that each word is conditionally independent on all but its $n-1$ preceding words. Such n-gram LM represents a form of *nonlocal parameterisation* and it is equivalent to a weighted finite-state automaton. Formally, rescoring the derivations in the hypergraph is equivalent to intersecting it with the automaton that represents the language model (Chiang, 2005; Kumar et al., 2006; Dyer et al., 2008).

Due to the complexity of the underlying search space, decoding is typically

[2]For a comprehensive survey refer to (Lopez, 2008).

done with heuristic search algorithms (Koehn et al., 2003; Chiang, 2007).[3] These algorithms are a form of approximate intersection and they operate by constructing only part of the space of derivations. They are guided by heuristics that attempt to predict high-scoring derivations while pruning low-scoring ones. Therefore, they lack formal guarantees.

In hierarchical models, translational equivalence is expressed by a synchronous grammar, which describes a polynomial number of reordering operators. Thus, intersection with the language model remains one of the biggest challenges. In phrase-based models, translations are produced from left to right, while the input is covered in arbitrary order under the constraint that each input word is translated exactly once. Ensuring this "non-overlapping constraint" requires an exponential number of states, thus, phrase-based models can be intractable even before intersection with the language model (Knight, 1999; Zaslavskiy et al., 2009).

In both cases, sampling has been somewhat neglected. Because the fully parameterised space of derivations is typically intractable, one cannot easily normalise the distribution defined by the hypergraph. Recall that normalisation requires summing over all the exponentially many derivations in the hypergraph. Unfortunately, one cannot easily resort to pruning either, because this typically introduces arbitrary bias to the distribution changing its priors. An alternative concerns the use of Markov Chain Monte Carlo (MCMC) methods, such as Gibbs sampling (Arun et al., 2009, 2010). However, the accuracy of a Gibbs sampler, and of MCMC methods in general, is hard to assess and formal guarantees are very loose.

This work addresses exact **optimisation** (decoding) and **sampling** in a unified framework by relying on a form of rejection sampling (Robert and Casella, 2004), a Monte Carlo (MC) method. In short, an intractable *goal*

[3]Although there are successful attempts to perform exact decoding under certain conditions (Chapter 3).

distribution, over which one wishes to perform inference, is *upperbounded* by tractable *proposal distributions*. A proposal represents a *relaxed* version of the complete space of weighted derivations, where relaxation happens with respect to the incorporation of the language model, and with respect to the non-overlapping constraint (relevant to phrase-based models only). These proposals give an optimistic view on the true model and allow for easier and faster search using standard dynamic programming techniques.

The approach introduced here is based on the OS* algorithm (Dymetman et al., 2012b,a), a technique inspired by rejection sampling. In rejection sampling, samples are drawn from a proposal distribution and accepted or rejected as a function of the mismatch between the proposal and the goal. This technique is adaptive in that rejected samples are used to motivate a refinement of the upperbound proposal that brings it closer to the goal, improving the rate of acceptance. Optimisation can be connected to an extreme form of sampling (Kirkpatrick et al., 1983; Dymetman et al., 2012b), thus the framework introduced here suits both exact optimisation and exact sampling. By **exact optimisation** we mean that the global maximum is found with a certificate of optimality. By **exact sampling** we mean that unbiased samples are independently drawn from the true distribution.

1.1 Hypotheses

Nonlocal parameterisation is one of the most important sources of complexity in SMT. For example, consider the costly intersection between a translation hypergraph and a target language model. Pruning turns out as the most popular way of handling the tractability issues that arise from nonlocal parameterisation. In sampling, Gibbs sampling offers a way around the need to represent an intractably large space of solutions, however, it comes with convergence properties hard to assess.

This thesis advocates for a different approach based on a more fundamental MC method, namely, rejection sampling. Rejection sampling offers strong guarantees, but it is generally intractable for large multidimensional discrete distributions. We achieve tractability in incorporating nonlocal parameterisation by avoiding the explicit intersection with the language model. We start with a much simpler (and optimistic) view of the LM distribution that forgets all context histories. This simpler proposal enables dynamic programming without pruning. We refine this proposal distribution incorporating selected histories on demand, a type of coarse-to-fine search strategy. In the case of phrase-based SMT, tractability without limits in distortion requires further relaxing the search space by removal of the non-overlapping constraints, which are then added on demand.

The following statements outline the main hypothesis of this work.

H1 A full intersection between a translation hypergraph (hierarchical or phrase-based) and a target language model aims at incorporating nonlocal parameterisation (the LM scores) and represents the goal distribution over which one wishes the perform inference. Explicitly performing this intersection is wasteful, in that most of the interactions arising from the nonlocal parameterisation are unlikely even on optimistic assessments using simpler upperbounds of the goal.

H2 Not only the space of n-grams that are likely to participate in high scoring derivations is much smaller than the full language model, but also they do not need to be incorporated everywhere in the translation hypergraph. Instead, they can be incorporated in very specific regions of this space, where edges have high marginal probabilities.

H3 In the context of phrase-based SMT, it is possible to dynamically manage the exponential number of states necessary to satisfy the non-overlapping constraint. Rather than arbitrarily stipulating a distortion limit, these constraints can be overlooked at first and added locally on demand throughout the search.

H4 A rejection sampler obtained for a sufficiently good nonlocal parameterisation can be used, with no further increase in complexity, to perform exact sampling from a model with longer-range dependencies, such as a higher-order n-gram language model.

1.2 Contributions of this thesis

This thesis makes a number of novel contributions to research with respect to exact optimisation and exact sampling for both phrase-based and hierarhical models of translation.

C1 An approach to exact inference in SMT based on the OS* algorithm. The main insight is to lower the complexity associated with the incorporation of nonlocal parameterisation, such as the language model component. We show that a very small subset of all n-grams in a language model needs to be incorporated before convergence (**H1**)

Chapter 4 introduces the general method;

Chapter 5 introduces the case of hierarchical phrase-based models;

Chapter 6 introduces the case of phrase-based models.

C2 An algorithm to selectively intersect a hypergraph and an automaton relying on edge marginals (Section 5.3.2). Although this algorithm is applied to inference in SMT, it is applicable to other problems, such as parsing, inference with high-order HMMs and other graphical models. We show that variable order n-grams can be incorporated only in high-probability regions of the hypergraph, rather than everywhere (**H2**).

C3 Tractable upperbounds on a phrase-based translation lattice without a distortion limit and a strategy for lazy incorporation of the non-overlapping constraint on demand (Section 6.2.2). We show that inference can be performed in the absence of a distortion limit without incorporating all the exponentially many constraints that prevent phrases from overlapping (**H3**). However, this is only true for very short sentences (up to 10 words). Nevertheless, these upperbounds are novel and can be potentially used with other forms of soft incorporation of non-overlapping constraints, such as recent work on Lagrangian relaxation has shown.

C4 We show that a rejection sampler designed to achieve a minimum acceptance rate for a certain nonlocal parameterisation (e.g. a 2-gram language model) can be used to produce samples from a model parameterised with wider-range dependencies (e.g. a 4-gram language model) at some small cost in acceptance rate (**H4**).

C5 An efficient computation of an upperbound on a language model distribution on a sentence basis (Section 4.2.2.1).

C6 An efficient procedure to tighten a language model upperbound which in turn leads to faster convergence (Section 4.2.2.2).

C7 A faster intersection between weighted context-free grammars and weighted automata relying on an incremental aspect of the algorithm introduced in Section 4.2 (Section 4.2.3).

1.3 Structure of the thesis

In Chapter 2, we revisit formal concepts and definitions that are necessary to understand the thesis. First we revisit algebraic operations over *weighted sets*, as well as the definition of *semiring* necessary to generalise several algorithms. Then we revisit concepts from *automata theory* and *formal languages*, namely, finite-state automata and context-free grammars, as well as their generalisations to multiple languages. A formal presentation of these machines is essential to the developments of this thesis. A uniform *graphical representation* of these formalism is also revisited. This *hypergraph* representation allows to unify important operations, such as weighted intersection, and inference algorithms, such as Inside-Outside. We also give a quick introduction to SMT (Section 2.4).

In Chapter 3, we survey related work on semiring-generalised *inference algorithms* and the state-of-the-art in terms of inference in SMT, including techniques based on *beam-search*, *cube-pruning*, *Gibbs sampling* and *Lagrangian relaxation*.

Chapters 4, 5 and 6 present this work's novel contributions. Chapter 4 introduces a novel framework for exact inference in SMT based on the OS* algorithm. We first revisit the OS*, then introduce an OS* approach to optimisation and sampling in SMT. Chapter 5 introduces the case of hierarchical phrase-based SMT with our new algorithm. We revisit in details the intersection between grammars and automata and describe our contributions towards obtaining faster and more compact intersections. Chapter 6 introduces the case of phrase-based SMT in which we also deal with the harder problem of decoding without a distortion limit, falling back to the original NP-complete formulation of phrase-based SMT.

Finally, Chapter 7 concludes this thesis with a summary of findings and a discussion about directions for future work including: speed ups in intersection and exact sampling with arbitrary nonlocal parameterisation, such as features that require a complete derivation in order to be assessed.

Chapter 2

Background

This chapter revisits concepts and definitions that are central to the developments discussed in this thesis. In Section 2.1, we revisit algebraic operations over *weighted sets*, as well as the definition of *semiring* necessary to generalise several inference algorithms. Then in Section 2.2, we revisit concepts from *automata theory* and *formal languages*, namely, finite-state automata and context-free grammars, as well as their generalisation to multiple languages. A uniform *graphical representation* of these formalism is revisited in Section 2.3. This *hypergraph* representation allows to unify important operations, such as weighted intersection, and inference algorithms, such as the Inside-Outside algorithm. Finally, Section 2.5 revisits an abstract formulation of SMT in terms of an algebra of weighted sets and relations.

2.1 Algebraic preliminaries

This section presents a formal definition of the most important structures and operations discussed in this thesis. We deliberately use a similar notation to those in (Goodman, 1998; Nederhof and Satta, 2008a; Mohri, 2009; Dyer, 2010).

2.1.1 Semirings

A semiring K is an algebraic structure denoted by a 5-tuple $\langle \mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1} \rangle$. It consists of a set $\mathbb{K}$ (e.g. $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$, $\{0, 1\}$, etc.) equipped with two binary operations. One operation, whose operator is $\oplus$, is called *addition*. The addition is commutative (i.e. $a \oplus b = b \oplus a$) and has identity $\bar{0}$ (i.e. $\bar{0} \oplus a = a \oplus \bar{0} = a$). The other operation, whose operator is $\otimes$, is called *multiplication*. The multiplication is associative (i.e. $(a \otimes b) \otimes c = a \otimes (b \otimes c)$) and has identity $\bar{1}$ (i.e. $\bar{1} \otimes a = a \otimes \bar{1} = a$).[1] Moreover, multiplication left and right *distributes* over addition (i.e. $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ and $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$). Finally, the addition identity $\bar{0}$ is the multiplicative annihilator (i.e. $\bar{0} \otimes a = a \otimes \bar{0} = \bar{0}$).

Table 2.1 lists a number of popular semirings, including different names by which they are known. The operator $\oplus_{\log}$ is such that $x \oplus_{\log} y = -\log(e^{-x} + e^{-y})$. A *commutative* semiring is such that the multiplication is commutative. The semirings in Table 2.1 are all commutative. An *idempotent* semiring is such that the addition is idempotent (i.e. $a \oplus a = a$). In Table 2.1, the binary, the artic and the tropical semirings are idempotent.

Semiring	$\mathbb{K}$	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$
Binary	$\{\text{true}, \text{false}\}$	$\vee$	$\wedge$	true	false
Counting	$\mathbb{N}$	$+$	$\times$	0	1
Sum-times (log)	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\oplus_{\log}$	$+$	$+\infty$	0
Max-times (artic)	$\mathbb{R} \cup \{-\infty\}$	max	$+$	$-\infty$	0
Min-times (tropical)	$\mathbb{R} \cup \{+\infty\}$	min	$+$	$+\infty$	0

Table 2.1: Example of semirings

Inference algorithms have different interpretations when operating under different semirings. Goodman (1999) introduces semiring parsing for probabilistic context-free grammars. Li and Eisner (2009) introduce semirings to compute first and second-order expectations using the Inside-Outside algorithm. Kumar et al. (2009) introduce a semiring named the "upper-envelop semiring" and show that Minimum Error Rate Training (MERT) (Och, 2003) is equivalent to running the Inside algorithm with that semiring.

[1]To avoid confusion with standard addition and multiplication, throughout the text we write $\oplus$-sum and $\otimes$-product meaning "sum using $\oplus$" and "product using $\otimes$", respectively.

2.1.2 Weighted sets

A weighted set generalises a standard set in that elements of the former are assigned weights. Examples of weighted sets are weighted finite-state automata and weighted context-free grammars. They compactly represent (possibly infinite) elements, such as the space of sentences in a language, while assigning them weights. The interpretation of these weights depends on the choice of semiring. For instance, they might represent the probability that a string is expected to be well-formed.

Formally, a weighted set $A = \langle X, w \rangle$ over semiring K is a pair of a (standard) set X and a weight function $w : X \to \mathbb{K}$ that assigns a weight to each element in X.

It is also convenient to define weighted sets of pairs of elements, where a pair expresses a form of input-output relation. This particular type of weighted set is called a *weighted binary relation*. Examples of weighted binary relations are weighted finite-state transducers and weighted synchronous context-free grammars. They compactly represent (possibly infinite) elements, such as the bilingual correspondences between two languages, while assigning them weights. As expected, the interpretation of the weights depends on the choice of semiring. For instance, they might represent the probability that an output string translates an input string.

Formally, a weighted binary relation $\mathcal{B} = \langle R \subseteq X \times Y, u \rangle$ is a weighted set over semiring K where items are mappings of elements of a domain X onto elements of a codomain Y with weight function $u : R \to \mathbb{K}$.

Sometimes a weighted set $S = \langle X, u \rangle$ will be treated as a weighted binary relation. In such cases we mean the relation $\mathcal{S} = \langle \{(x, x) : x \in X\}, w(x, x) = u(x) \rangle$. Following Dyer (2010), we will call $\mathcal{S}$ an identity-transducer and the word *transducer* will refer to either a Weighted Finite-State Transducer (wFST) or a Weighted Synchronous Context-Free Grammar (wSCFG), depending on the context.[2]

2.1.3 Operations over weighted sets

This section revisits important operations over weighted sets. We discuss the intersection between weighted sets in Section 2.1.3.1. In Section 2.1.3.2, we define the difference between a weighted set and an unweighted set. The last two operations concern weighted binary relations. In Section 2.1.3.3, we discuss the composition of two weighted binary relations. Finally, we define the weighted projection of a binary relation onto one of its dimensions (Section 2.1.3.4).

2.1.3.1 Intersection

Intuitively, the intersection between weighted sets is very similar to the intersection between standard (unweighted) sets. In the standard case, given two sets, the intersection retains only the elements that are common to both input sets. In the weighted case, additionally, the elements retained by the intersection are weighted by the product of their original weights in each input set.

Formally, given the weighted sets $\langle A, u\rangle$ and $\langle B, v\rangle$ over semiring K, the weighted intersection $\langle A, u\rangle \cap \langle B, v\rangle = \langle C, w : C \to \mathbb{K}\rangle$ is such that: $C = A \cap B = \{x : x \in A \wedge x \in B\}$ and

$$w(x) = \begin{cases} u(x) \otimes v(x) & \text{if } x \in A \wedge x \in B \\ \bar{0} & \text{otherwise.} \end{cases}$$

That is, if an element x is common to A and B, x is retained in C and it is weighted by the $\otimes$-product $u(x) \otimes v(x)$. Any other element weights $\bar{0}$.

[2]For clarity, we use different typesets to refer to weighted sets and weighted binary relations. Moreover, if a weighted set S has been defined, $\mathcal{S}$ will refer to its identity-transducer.

2.1.3.2 Difference

Intuitively, the difference between two sets $A - B$ is an operation that removes from A elements that belong to B. For weighted sets over semiring K, this operation would require K to define a negation operator $\ominus$, however, recall from the definition in Section 2.1.1 that a semiring requires only two operators, namely, $\oplus$ and $\otimes$.[3]

Formally, the difference operation can be defined for a weighted set $\langle A \subseteq X, u\rangle$ and an unweighted set $B \subseteq X$. The result is the weighted set $\langle D, w\rangle$ such that $D = \{x \in X : x \in A \wedge x \notin B\}$ and

$$w(x) = \begin{cases} u(x) & \text{if } x \in A \wedge x \notin B \\ \bar{0} & \text{otherwise.} \end{cases}$$

The difference can also be defined in terms of the intersection between two weighted sets, namely, $\langle A \subseteq X, u\rangle$ and $\langle C = X \setminus B, v\rangle$, where the latter is called the *complement* of B in X. The complement contains the elements of X that do not belong to B, that is, $C = \{x \in X : x \notin B\}$. These elements are weighted so that $v(x) = \bar{1}$ if $x \in C$, and $v(x) = \bar{0}$ otherwise. Therefore, one can write the following equivalences

$$\langle A \subseteq X, u\rangle - B = \langle A \subseteq X, u\rangle \cap \langle C = X \setminus B, v\rangle = \langle A \setminus B \subseteq X, w\rangle$$

where $w(x) = u(x) \otimes v(x)$.

2.1.3.3 Composition

The weighted composition is the generalisation of weighted intersection for weighted binary relations. Consider two weighted binary relations $\mathcal{A} = \langle R_A \subseteq X \times Y, u\rangle$ and $\mathcal{B} = \langle R_B \subseteq Y \times Z, v\rangle$, where the domain of $\mathcal{B}$ (its

[3] Note that for some semirings the negation operator does make sense, e.g. the Boolean semiring.

input) is the codomain of $\mathcal{A}$ (its output). Intuitively, the weighted composition operates over the two sets abstracting away the elements of Y. That is, the composed relation maps directly from X to Z implicitly *pivoting* through elements in Y.

Formally, the weighted composition $\mathcal{A} \circ \mathcal{B}$ defines a binary relation $\mathcal{C} = \langle R_C \subseteq X \times Z, w \rangle$ such that $C = \{(x,z) | \exists y \in Y : (x,y) \in R_A \wedge (y,z) \in R_B\}$ and

$$w(x,z) = \begin{cases} \bigoplus\limits_{\substack{y \in Y \\ (x,y) \in R_A \\ (y,z) \in R_B}} u(x,y) \otimes v(y,z) & \text{if } (x,y) \in R_A \wedge (y,z) \in R_B \\ \bar{0} & \text{otherwise.} \end{cases}$$

That is, the elements $y \in Y$ for which $(x,y) \in R_A$ and $(y,z) \in R_B$ are pivots that enable the direct mapping (x,z). The pair (x,z) is weighted by the $\oplus$-sum over the $\otimes$-product of the pairs involving the pivot elements. In the context of probabilistic inference, this operation can be thought of as marginalising out a latent variable.

2.1.3.4 Projection

Given a weighted binary relation $\mathcal{B} = \langle R \subseteq X \times Y, u \rangle$ over semiring K, the projection of $\mathcal{B}$ onto its codomain (also denoted *output projection*) is the weighted set $\mathcal{B} \downarrow = \langle R, u \rangle \downarrow = \langle A, w \rangle$, such that $A = \{y | \exists x \in X : (x,y) \in R\}$ and $w(y) = \bigoplus_{x \in X:(x,y) \in R} u(x,y)$.

The projection is itself a weighted set, where the weight of an item y is the $\oplus$-sum of $u(x,y)$ over all possible values of the input item x. In the context of probabilistic inference, this operation can be thought of as summing over all possible values X of a random variable.

It is then straightforward to define the input projection, or projection onto the domain, which is denoted by $\mathcal{B} \uparrow = \langle R, u \rangle \uparrow$, and $\oplus$-summation happens over $y \in Y : (x, y) \in R$.

2.2 Automata and grammars

In this section we revisit two tractable representations of weighted sets and their generalisations to weighted binary relations. There are several equivalent ways in which this presentation could be done without altering the expressiveness of the machinery discussed. We opt to mostly follow the presentation in (Dyer, 2010) due to its algebraic groundings.

2.2.1 Weighted finite-state automata

Informally, a *Weighted Finite-State Automaton* (wFSA) is made of a set of states connected by transitions, which in turn recognise (or accept) symbols of a given vocabulary (or alphabet), assigning them weights. A valid string, that is, a string recognised by the Weighted Finite-State Automaton (wFSA), is such that it can be associated to at least one sequence of transitions departing from an initial state of the wFSA and arriving at a final state of the wFSA. We call such sequence of transitions a (valid) path in the wFSA. The total weight assigned to any given string is the sum over all paths recognising that string. Additionally, initial and final states are themselves weighted, and those weights contribute to the weight of every valid path.

Formally, a weighted finite-state automaton over semiring K is defined as a 5-tuple $A = \langle \Sigma, Q, \langle I, \lambda \rangle, \langle F, \rho \rangle, \langle E, w \rangle \rangle$ where:

- Σ is a finite set of terminal symbols, also known as alphabet or vocabulary;
- Q is a finite set of states;
- $\langle I, \lambda \rangle$ is a finite weighted set of initial states $I \subseteq Q$ with weight function $\lambda : I \to \mathbb{K}$;

- $\langle F, \rho \rangle$ is a finite weighted set of accepting (final) states $F \subseteq Q$ with weight function $\rho : F \to \mathbb{K}$;
- $\langle E, w \rangle$ is a finite weighted set of transitions (or edges) $E \subseteq (Q \times \Sigma \times Q)$ with weight function $w : E \to \mathbb{K}$.

A transition $e \in E$ connects two states in Q, one being its origin state, denoted by $p[e]$, another being its destination state, denoted by $n[e]$. Moreover it recognises a symbol from Σ, denoted by $i[e]$ and often called an *input* symbol, and it carries a weight in $\mathbb{K}$, denoted by $w(e)$.

A path in a wFSA is a sequence of l *adjacent* transitions denoted by $\pi = \langle e_1, e_2, \dots, e_l \rangle$, where $e_i \in E$, and by adjacency we mean that $n[e_i] = p[e_{i+1}]$ for $i \in [1, l-1]$. A path can be inspected in terms of (1) its first state $p[\pi] = p[e_1]$, that is, the origin state of its first transition; (2) its last state $n[\pi] = n[e_l]$, that is, the destination state of its last transition; (3) the string $i[\pi] = \langle i[e_1], i[e_2], \dots, i[e_l] \rangle \in \Sigma^*$ it recognises, that is, the concatenation of the input symbols of the transitions along the path; and finally, (4) its weight $w[\pi] = \bigotimes_{i=1}^{l} w(e_i)$, that is, the $\otimes$-product of the weights of the transitions along the path.

To understand the set of strings over the vocabulary Σ defined by a wFSA, its *language*, let $P(o, d)$ be the set of all paths from a state $o \in Q$ to a state $d \in Q$. We can then restrict $P(o, d)$ to paths that recognise a specific string $x \in \Sigma^*$, and denote it by $P(o, x, d) = \{\pi \in P(o, d) : i[\pi] = x\}$. Finally this set can be generalised to account for a set O of origin states and a set D of destination states, that is, $P(O \subseteq Q, x, D \subseteq Q) = \bigcup_{o \in O \wedge d \in D} P(o, x, d)$.

With these definitions we have enough to formalise the set of strings that represents the language $L(A) \subseteq \Sigma^*$ of the automaton A:

$$L(A) = \{x \in \Sigma^* : P(I, x, F) \neq \emptyset\} \subseteq \Sigma^*$$

these are all strings in Σ^* for which there is at least one path in A connecting one of its initial states to one of its final states. A wFSA therefore defines a weighted set $\langle L(A), u\rangle$ in K with weight function given by Equation 2.1.

$$u(x) = \begin{cases} \bar{0} & \text{if } P(I,x,F) = \emptyset \\ \bigoplus_{\pi \in P(I,x,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi]) & \text{otherwise.} \end{cases} \tag{2.1}$$

Note in Equation 2.1 that when $P(I,x,F) = \emptyset$, there is no *valid* path in A that recognises x, and the weight assigned to x is, thus, $\bar{0}$. Otherwise, the weight assigned to x is the $\oplus$-sum over *all paths* that recognise x. Also note that the weight of a valid path is $\otimes$-multiplied by the weights of its initial and its final state.

2.2.2 Weighted finite-state transducers

A *Weighted Finite-State Transducer* (wFST) generalises a wFSA to jointly recognise strings from multiple vocabularies. One can think of vocabularies as different dimensions, and wFSTs the multi-dimensional versions of wFSAs, where strings are recognised in parallel across dimensions. In this thesis we will focus on bilingual wFSTs, where one vocabulary concerns an input (or source) language, and will be referred to as the input (or source) vocabulary, and another concerns an output (or target) language, being referred to as the output (or target) vocabulary.

Formally, a weighted finite-state transducer over semiring K is defined as a 6-tuple $\mathcal{T} = \langle \Sigma, \Delta, Q, \langle I, \lambda\rangle, \langle F, \rho\rangle, \langle E, w\rangle\rangle$ where we extend the definition in Section 2.2.1 to account for an additional set of terminal symbols Δ, the output vocabulary, and change $\langle E, w\rangle$ to the finite weighted binary relation $E \subseteq (Q \times \Sigma \times \Delta \times Q)$ with weight function $w : E \to \mathbb{K}$.

A transition $e \in E$ describes a jump from an origin state to a destination state, it now takes simultaneously two symbols, an input symbol from Σ

denoted by $i[e]$, and an output symbol from Δ denoted by $o[e]$, and it is associated to a weight $w(e)$ in $\mathbb{K}$.

Just like in the wFSA case, a path π is defined as a sequence of l adjacent transitions, and it can be inspected in terms of $p[\pi]$, $n[\pi]$, $i[\pi]$, and $w[\pi]$ as defined in Section 2.2.1. Additionally, the path recognises (or produces) the output string $o[\pi] = \langle o[e_1], o[e_2], \ldots, o[e_l]\rangle$, which is made of the concatenation of the output symbols at each transition along the path.

We can also define $P(o, x, y, d) = \{\pi \in P(o, d) : i[\pi] = x \wedge o[\pi] = y\}$, where $o \in Q$, $d \in Q$, $x \in \Sigma^*$ and $y \in \Delta^*$, the set of all paths from an origin state o to a destination state d that accept the pair of strings (x, y). The generalisation $P(O \subseteq Q, x, y, D \subseteq Q)$ follows straightforwardly.

With these definitions we have enough to formalise the set of *pairs of strings* that represents the *language* $L(\mathcal{T}) \subseteq \Sigma^* \times \Delta^*$ of the transducer $\mathcal{T}$:[4]

$$L(\mathcal{T}) = \{(x, y)|x \in \Sigma^*, y \in \Delta^* : P(I, x, y, F) \neq \emptyset\} \subseteq \Sigma^* \times \Delta^*$$

these are all pairs of strings in $\Sigma^* \times \Delta^*$ for which there is at least one path in $\mathcal{T}$ connecting one of its initial states to one of its final states. A wFST therefore defines a weighted binary relation $\langle L(\mathcal{T}), u\rangle$ in K with weight function given by Equation 2.2.

$$u(x, y) = \begin{cases} \bar{0} & \text{if } P(I, x, y, F) = \emptyset \\ \displaystyle\bigoplus_{\pi \in P(I, x, y, F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi]) & \text{otherwise.} \end{cases} \tag{2.2}$$

Note that the weight function is defined over pairs of strings. To reason about output strings independently from the input strings, one needs to marginalise out, or $\oplus$-sum over, the input strings that project onto the same output string, and similarly vice-versa, an operation called output (or

[4]With certain abuse of language, we call the set of *pairs of strings* recognised by a wFST its *language*, in analogy to the terminology used for wFSAs.

input) projection (see Section 2.1.3.4). For a detailed overview on finite-state transducers and their operations, including algorithms, refer to (Mohri, 2009).

2.2.3 Weighted context-free grammars

Informally, a *Weighted Context-Free Grammar* (wCFG) is a rewrite system made of a set of rules defined over a set of variables and an alphabet of terminal symbols. The rules describe how variables can be rewritten, one at a time from left to right, regardless of context, into sequences of variables and symbols of the alphabet. A sequence of rewrite operations that leads to all variables being expanded yielding a string of symbols of the alphabet is known as a derivation. Rules are associated to weights and the total weight of a derivation is the product of its rules. Finally, the total weight of a string recognised (or produced) by the grammar is the sum over all derivations that yield that string. It is intuitive to think of Weighted Context-Free Grammars (wCFGs) bottom-up, that is, we start with a string of terminal symbols and group blocks within this string under variables, each block being directly below a single variable. Sequences of variables and blocks of strings get combined under other variables, creating a tree structure, up to the point that a single variable is proven at the top of the tree. The tree itself represents a derivation and the top variable is also called root.

Formally, a weighted context-free grammar over semiring K is defined as a 4-tuple $G = \langle \Sigma, V, \langle S, \sigma \rangle, \langle R, \nu \rangle \rangle$ where:

- Σ is a finite set of terminal symbols (the alphabet);
- V is a finite set of nonterminal symbols (or variables), and $\Sigma \cap V = \emptyset$;
- $\langle S, \sigma \rangle$ is a finite weighted set of start (or root) symbols $S \subseteq V$ with weight function $\sigma : S \to \mathbb{K}$;

- $\langle R, \nu\rangle$ is a finite weighted set of context-free rewrite rules (or productions). More specifically, a weighted binary relation, where $R \subseteq V \times (V \cup \Sigma)^*$ with weight function $\nu : R \to \mathbb{K}$;

A rule $r \in R$ can be inspected in terms of (1) its left-hand side, a single nonterminal symbol, denoted by $\text{LHS}[r] \in V$; (2) its right-hand side (or yield), a sequence of terminal and nonterminal symbols, denoted by $\text{RHS}_i[r] \in (V \cup \Sigma)^*$;[5] (3) the number of nonterminal symbols in its yield, its arity, denoted by $a[r]$; and finally, (4) its weight $\nu(r)$ in $\mathbb{K}$. A rule r can be written $\text{LHS}[r] \underset{\nu_r}{\longrightarrow} \text{RHS}_i[r]$, where the weight ν_r might be omitted if not relevant to the point being made.

To understand how rules are applied consider two sequences u and v of terminals and nonterminals symbols, that is, $u, v \in (V \cup \Sigma^*)$. We then say that u yields v if, and only if, there exists a rule $\alpha \to \beta$ in R, therefore, $\alpha \in V$ and $\beta \in (V \cup \Sigma^*)$, such that $u = u_1\alpha u_2$ and $v = u_1\beta u_2$, where $u_1 \in \Sigma^*$ is a terminal prefix to nonterminal α, and u_2 is a sequence in $(V \cup \Sigma^*)$. Note that because u_1 must be a sequence of *terminal* symbols, the rule application rewrites the left-most nonterminal α in u, thus each derivation can only produce one tree. In short, we write $u \xRightarrow{\alpha \to \beta} v$ and read u yields v by application of rule $\alpha \to \beta$.

A derivation $\delta = \langle r_1, r_2, \ldots, r_l\rangle \in R$ is made of l successive rule applications, resulting in a string of terminals, that is, $\text{LHS}[r_1] \overset{r_1}{\Rightarrow} (V \cup \Sigma)^* \overset{r_2}{\Rightarrow} \cdots \overset{r_l}{\Rightarrow} \Sigma^*$, or in short $\text{LHS}[r_1] \overset{\delta}{\Rightarrow} \Sigma^*$. We can inspect a derivation in terms of (1) its root symbol $\text{LHS}[\delta]$, which is the left-hand side of its first rule $\text{LHS}[r_1]$; (2) its yield $\text{RHS}_i[\delta] \in \Sigma^*$, which is the string resulting of all l rule applications, starting from the root; and finally, (3) its weight $\nu[\delta] = \bigotimes_{i=1}^{l} \nu(r_i)$, obtained by $\otimes$-multiplying the weights of the rules in δ.

Similarly to what was done with wFSAs, we can talk about special groups

[5]The index i in $\text{RHS}_i[r]$ stands for *input*. Its need will become obvious in Section 2.2.4, when wCFGs are generalised into wSCFGs.

of derivations. The set of all derivations rooted at nonterminal $q \in V$ is written $D(q) = \{\delta \in R^* : \mathrm{LHS}[\delta] = q\}$. Thus, $D(q, x) = \{\delta \in R^* : \mathrm{LHS}[\delta] = q \wedge \mathrm{RHS}_i[\delta] = x\}$, where $q \in V$, is the set of all derivations rooted at q that yield the string $x \in \Sigma^*$. The generalisation of $D(q, x)$ to a set of roots $Q \subseteq V$ is straightforward: $D(Q, x) = \bigcup_{q \in Q} D(q, x)$.

We can now formalise the set of strings in the language $L(G)$ recognised by the grammar G:

$$L(G) = \{x \in \Sigma^* : D(S, x) \neq \emptyset\}$$

these are all strings in Σ^* for which there is at least one derivation in R^* rooted at a start symbol in S. A wCFG therefore defines a weighted set $\langle L(G), w\rangle$ in K with weight function given by Equation 2.3.

$$w(x) = \begin{cases} \bar{0} & \text{if } D(S, x) = \emptyset \\ \displaystyle\bigoplus_{\delta \in D(S,x)} \sigma(\mathrm{LHS}[\delta]) \otimes \nu[\delta] & \text{otherwise.} \end{cases} \tag{2.3}$$

Note in Equation 2.3 that when $D(S, x) = \emptyset$, there is no derivation in G that recognises x, and the weight assigned to x is, thus, $\bar{0}$. Otherwise, the weight assigned to x is the $\oplus$-sum over *all derivations* that recognise x. Also note that the weight of a valid path is $\otimes$-multiplied by the weight of its start symbol.

2.2.4 Weighted synchronous context-free grammars

In the same way that wFSTs generalise wFSAs, a *Weighted Synchronous Context-Free Grammar* (wSCFG) generalises a wCFG to jointly recognise strings from multiple vocabularies. Informally, one can think of a wSCFG as two CFGs, one over an input vocabulary and the other over an output vocabulary, that share a set of nonterminal variables, and where a production in either one of them is paired with a production in the other with a one-to-one correspondence between nonterminals (hence the term *synchronous*).

Numerous formalism related to wSCFGs have been proposed (Lewis and Stearns, 1968; Aho and Ullman, 1969; Wu, 1995, 1997; Melamed, 2003; Chiang, 2005, 2007). Lewis and Stearns (1968) were probably the first to introduce a bilingual generalisation of CFGs, which they called *Syntax-Directed Transducer* (STD). Wu (1995, 1997) introduces a restrict case of the more general wSCFGs, the *Inversion Transduction Grammar* (ITG), where nonterminals in output rules either preserve or invert the orientation of the input. Chiang (2005) renamed the syntax-directed transducers to *synchronous context-free grammars*, a term which probably better conveys what they are.[6] Melamed (2003) introduced Multitext Grammars (MTGs), a generalisation of wSCFGs which handles an arbitrary number of dimensions.

Aho and Ullman (1969) formalised synchronous rules as $\alpha \rightarrow \langle \beta, \gamma, \Pi \rangle$, where α is a nonterminal symbol from a shared set of variables, β is a sequence of nonterminals and input terminals, γ is a sequence of nonterminals and output terminals, and Π is a permutation function $\Pi : \{1, a[\beta]\} \rightarrow \{1, a[\beta]\}$ that defines the one-to-one mapping between variables in β and variables in γ (β and γ always have the same number of variables). In other words, the i^{th} variable in β *corresponds* to the $\Pi(i)^{th}$ variable in γ. In (Chiang, 2005), the permutation function $\sim$ plays exactly the same role as Π. Dyer (2010) uses a permutation function only implicitly. Instead, he defines a vocabulary of indexed gaps which express the one-to-one mapping between variables in the input right-hand side and the gaps in the output right-hand side.

Formally, a weighted synchronous context-free grammar over semiring K is defined as a 5-tuple $\mathcal{G} = \langle \Sigma, \Delta, V, \langle S, \sigma \rangle, \langle R, \nu \rangle \rangle$ where we extend the definition in Section 2.2.3 to account for an additional set of terminal symbols Δ, the output vocabulary, and a weighted set $\langle R, \nu \rangle$ of *synchronous rules*.

[6] Chiang (2007) noted that a wSCFG constrained to handling two nonterminals on the righ-hand side of its synchronous rules is equivalent to an ITG. This constraint facilitates unsupervised learning of wSCFGs as well as decoding.

The set V of nonterminal symbols is shared between Σ and Δ, and $\Sigma \cap V = \Delta \cap V = \emptyset$. A synchronous rule $\alpha \rightarrow \langle \beta, \gamma \rangle$ in R expresses a relation between a common left-hand side nonterminal $\alpha \in V$ and a pair $\langle \beta, \gamma \rangle$ of rewrite rules (the input and the output right-hand side of the rule). The input $\beta \in (V \cup \Sigma)^*$ is a sequence of nonterminals and input terminals. The output $\gamma \in (\{\boxed{1}, \boxed{2}, \ldots\} \cup \Delta)^*$ is a sequence of *indexed gaps* and output terminals, such that the number $a[\gamma]$ of indexed gaps in γ is equal to the number $a[\beta]$ of variables in β. These gaps are positions that range from 1 to $a[\gamma]$ with no repetition, and they may show in γ in any order. Moreover, they realise a one-to-one *correspondence* to nonterminals in β. Dyer (2010) introduced this formulation and named it a *gap correspondence constraint*, which implicitly shares the role of the permutation function in (Aho and Ullman, 1969; Chiang, 2005). A rule $r \in R$, where $R \subseteq V \times (V \cup \Sigma)^* \times (\{\boxed{1}, \boxed{2}, \ldots\} \cup \Sigma)^*$, can also be written $\text{LHS}[r] \xrightarrow[\nu_r]{} \langle \text{RHS}_i[r], \text{RHS}_o[r] \rangle$, where $\text{RHS}_o[r] \in (\{\boxed{1}, \boxed{2}, \ldots\} \cup \Delta)^*$ represents the output right-hand side (or output yield).

To understand rule application, first consider the function

$$Y_k : (\{\boxed{1}, \boxed{2}, \ldots\} \cup \Delta)^* \rightarrow (\{\boxed{1}, \boxed{2}, \ldots\} \cup \Delta)^*$$

that takes a sequence of indexed gaps and output terminals, and adds k to the index of every gap, for example $Y_{-1}(a\ \boxed{2}\ b) = a\ \boxed{1}\ b$. Then for $u, v \in (V \cup \Sigma)^*$ and $x, y \in (\{\boxed{1}, \boxed{2}, \ldots\} \cup \Delta)^*$, we say $\langle u, x \rangle$ yields $\langle v, y \rangle$, both pairs observing the correspondence constraints, if, and only if, there exists a rule $\alpha \rightarrow \langle \beta, \gamma \rangle$ in R such that $\langle u, x \rangle = \langle u_1 \alpha u_2, x_1 \boxed{1} x_2 \rangle$ and $\langle v, y \rangle = \langle u_1 \beta u_2, Y_{a[\beta]-1}(x_1) \gamma Y_{a[\beta]-1}(x_2) \rangle$, where $u_1 \in \Sigma^*$, $u_2 \in (V \cup \Sigma)^*$, and $x_1, x_2 \in (\{\boxed{1}, \boxed{2}, \ldots\} \cup \Delta)^*$. Note that u_1 is a sequence of input terminals, that is, α is the left-most variable in u, and as a consequence, replacing α by β in u requires replacing $\boxed{1}$ by γ in x. Moreover, because rule application always substitutes $\boxed{1}$, every indexed gap in x_1 and x_2 can only refer to a variable to the right of α in u. If the rule rewrites α (one nonterminal) using terminals only (i.e. $a[\beta] = 0$), the indexes must be reduced by 1 in x_1 and in x_2 (and y is one nonterminal shorter than x). If α is rewritten using a sequence containing one nonterminal (i.e. $a[\beta] = 1$), then the indexes will remain unchanged ($a[x] = a[y]$). Finally, if the rewrite introduces a sequence with $m + 1$ nonterminals (i.e. $a[\beta] = m + 1$) and $m > 0$, the indexes must be increased by m (y is m nonterminals longer than x). The function $Y_{a[\beta]-1}$ compactly updates the indexed gaps maintaining $\langle v, y\rangle$ well-formed.

A derivation $\delta = \langle r_1, r_2, \ldots, r_l\rangle \in R$ is made of l successive rule applications, resulting in a pair of terminal strings, that is, $\text{LHS}[r_1] \overset{r_1}{\Rightarrow} \langle (V \cup \Sigma)^*, (\{\boxed{1}, \boxed{2}, \ldots\} \cup \Delta)^*\rangle \overset{r_2}{\Rightarrow} \cdots \overset{r_l}{\Rightarrow} \langle \Sigma^*, \Delta^*\rangle$, or in short $\text{LHS}[r_1] \overset{\delta}{\Rightarrow} \langle \Sigma^*, \Delta^*\rangle$. In addition to what was defined in Section 2.2.3, we can inspect a derivation in terms of the output string it recognises (or produces), denoted by $\text{RHS}_o[\delta] \in \Delta^*$. We can then extend $D(q, x)$ to $D(q, x, y) = \{\delta \in R^* : \text{LHS}[\delta] = q \wedge \text{RHS}_i[\delta] = x \wedge \text{RHS}_o[\delta] = y\}$, which is the set of all derivations rooted at $q \in V$ that yield the pair of strings $\langle x \in \Sigma^*, y \in \Delta^*\rangle$. The generalisation to $D(Q \subseteq V, x, y) = \bigcup_{q \in Q} D(q, x, y)$ follows straightforwardly.

We can now formalise the set of pairs of strings in the language $L(\mathcal{G})$ recognised by the synchronous grammar $\mathcal{G}$:

$$L(\mathcal{G}) = \{(x, y) \in \Sigma^* \times \Delta^* : D(S, x, y) \neq \emptyset\}$$

these are all pairs of strings in $\Sigma^* \times \Delta^*$ for which there is at least one derivation in R^* rooted at a start symbol in S. A wSCFG therefore defines a weighted binary relation $\langle L(\mathcal{G}), w\rangle$ in K with weight function given by Equation 2.4.

$$w(x, y) = \begin{cases} \bar{0} & \text{if } D(S, x, y) = \emptyset \\ \displaystyle\bigoplus_{\delta \in D(S, x, y)} \sigma(\text{LHS}[\delta]) \otimes \nu[\delta] & \text{otherwise.} \end{cases} \tag{2.4}$$

Just like in the case of wFSTs, the weight function is defined over pairs of strings and to reason about output strings independently one needs to $\oplus$-sum over the input strings that project onto the same output string.

2.3 Hypergraphs

Informally, hypergraphs are generalisation of graphs where hyperedges — the generalisation of edges — have multiple origins and multiple destinations. They have been widely used to compactly represent weighted sets and relations — exponentially many items can be represented in polynomial space. They are compatible with numerous formalism from automata theory and formal languages.

Formally, a weighted directed hypergraph (Gallo et al., 1993) is a pair $\langle\langle V, \phi\rangle, \langle E, w\rangle\rangle$, where $\langle V, \phi\rangle$ is a weighted set of vertices (or nodes) and $\phi : V \to \mathbb{K}$, and $\langle E, w\rangle$ is a weighted set of hyperedges and $w : E \subseteq (V^* \times V^*) \to \mathbb{K}$. A hyperedge $e \in E$ is an ordered pair $(H \subseteq V^*, T \subseteq V^*)$ which relates two lists of vertices, the head H, also denoted $h[e]$, and the tail T, also denoted $t[e]$.[7] A hyperedge is directed, vertices in the tail precede vertices in the head.

In Natural Language Processing (NLP), particularly, a specific class of directed hypergraphs, the backward hypergraphs (or simply B-hypergraphs), is widely used to represent in a unified framework finite-state automata and context-free grammars, as well as their multidimensional generalisations. A B-hypergraph $\langle\langle V, \phi\rangle, \langle E, w\rangle\rangle$ is a directed hypergraph in which all hyperedges are backward edges (B-edges). A B-edge is a hyperedge headed by a single node, that is, $E \subseteq V \times V^*$, thus $|h[e]| = 1$ for $e \in E$.

Let $H = \langle\langle V', \phi\rangle, \langle E, w\rangle\rangle$ and $G = \langle\Sigma, V, \langle S, \sigma\rangle, \langle R, \nu\rangle\rangle$ be a B-hypergaph and a wCFG, respectively. With (1) conveniently chosen $\langle V', \phi\rangle$, such that $V' = \Sigma \cup V$, and $\phi(q) = \sigma(q)$ if $q \in S \subseteq V$, and $\bar{1}$ otherwise; also (2) by choosing $\langle E, w\rangle$, so that $E \subseteq V \times (\Sigma \cup V)^*$, thus an edge $e \in E$ is an ordered pair $e = \langle v_0, \langle v_1, \ldots, v_n\rangle\rangle$, where $h[e] = v_0 \in V$ is the head, and $t[e] = \langle v_1, \ldots, v_n\rangle$ is the tail; and finally, (3) by choosing w so that $w(e) =$

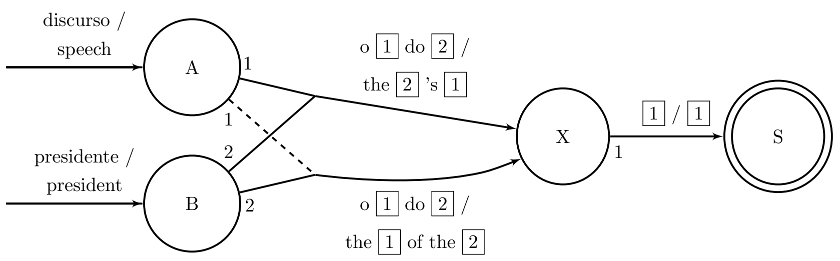
[7]In (Gallo et al., 1993), H and T are sets of vertices, however, it is common to extend that definition and treat H and T as lists or ordered multisets.

$\nu(h[e] \to t[e])$, we can see that a B-hypergraph is completely equivalent to a wCFG. Nodes heading edges are equivalent to nonterminals, $V = \{v \in V' : \exists e \in E, h[e] = v\}$, and nodes that do not head edges are equivalent to terminals, $\Sigma = \{v \in V' : \nexists e \in E, h[e] = v\}$. It is also convenient to have a single goal node v_g in the hypergraph, such that $\{e \in E : v_g \in t[e]\} = \emptyset$, which is straightforward to do.

Observe that B-hypergraphs where $|t[e]| = 1$ for every edge $e \in E$ are equivalent to directed graphs used to represent wFSAs (Dyer, 2010; Rush et al., 2013). This is a consequence of the fact that regular languages, those defined by finite-state automata, are included in the set of context-free languages (Hopcroft and Ullman, 1969, 1979). This means that hypergraphs make a very useful tool to handle wFSAs and wCFGs more uniformly and to have unified algorithms for both classes of machinery.

The B-edge e is equivalent to a context-freee rewrite rule r, where $w(e) = \nu(r)$, $h[e] = \text{LHS}[r]$ and $t[e] = \text{RHS}[r]$. Moreover for multidimensional B-hypergraphs — a generalisation equivalent to a wSCFG — let us also write $i[e] = \text{RHS}_i[r]$ and $o[e] = \text{RHS}_o[r]$. Figure 2.1a illustrates the hypergraph that is equivalent to the SCFG in Figure 2.1b. In Figure 2.1a, terminal nodes are omitted, instead, they are shown as labels on edges. The forward slash separates the input dimension from the output dimension. The double circle represents the goal node. Finally, a boxed number indicates the position of a corresponding nonterminal node in the tail of the edge.

To relate to the terminology previously used with wCFGs, vertices (or nodes) are equivalent to nonterminals, goal vertices are equivalent to start symbols, and hyperedges are equivalent to rewrite rules. Also connecting with the terminology used for wFSAs, states become vertices, final states are goal vertices and edges are transitions linking one origin state (the tail of the edge) to one destination state (the head of the edge). From now on, let us refer to B-hypergraphs simply as hypergraphs. Similarly, let us refer


discurso / speech
presidente / president
A
B
X
S
1
1
2
2
1
o $\boxed{1}$ do $\boxed{2}$ / the $\boxed{2}$'s $\boxed{1}$
o $\boxed{1}$ do $\boxed{2}$ / the $\boxed{1}$ of the $\boxed{2}$
$\boxed{1}$ / $\boxed{1}$

(a) Hypergraph

LHS	RHS_i	RHS_o
S →	X	$\boxed{1}$
X →	o A do B	the $\boxed{2}$'s $\boxed{1}$
X →	o A do B	the $\boxed{1}$ of the $\boxed{2}$
A →	discurso	speech
B →	presidente	president

(b) Synchronous rules

Figure 2.1: B-hypergraphs and context-free grammars

to hyperedges simply as edges. The terms vertex, node and nonterminal will be used as synonymous; also the terms edge and rule, and the terms goal node and root. A path in a hypergraph will be also called a derivation. Finally, for a node q, the *backward star* of q is the set of incoming edges to q, $BS(q) = \{e \in E : h[e] = q\}$, and the *forward star* of q is the set of outgoing edges from q, $FS(q) = \{e \in E : q \in t[e]\}$ (Gallo et al., 1993). In grammar parlance, $BS(q)$ is the set of all rules whose left-hand side is the nonterminal q, and $FS(q)$ is the set of all rules that use q in their right-hand sides.

2.4 Statistical machine translation

The task of producing a translation for a string $\mathbf{x}$ over an input vocabulary is typically associated to finding the best derivation $\mathbf{d}^*$ compatible with the input under a linear model. Where a derivation is a structured output that represents a sequence of steps that covers the input producing a translation.

This processes, also known as decoding, is illustrated in Equation 2.5.

$$\mathbf{d}^* = \operatorname*{argmax}_{\mathbf{d} \in \mathcal{D}(\mathbf{x})} f(\mathbf{d}, \Lambda) \tag{2.5}$$

The set $\mathcal{D}$, also denoted $\mathcal{D}(\mathbf{x})$ to stress the dependency on $\mathbf{x}$, is the space of all derivation compatible with $\mathbf{x}$ and supported by a model of translational equivalences. The function $f(\mathbf{d}, \Lambda) = \Lambda \cdot \mathbf{H}(\mathbf{d})$, also denoted $f(\mathbf{d})$ for simplicity, is a linear parameterisation of the model. It assigns a real-valued score (or weight) to every derivation $\mathbf{d} \in \mathcal{D}$. The score of a derivation does not need to have a probabilistic interpretation, however, $p_\Lambda(\mathbf{d}) = \exp\{f(\mathbf{d}, \Lambda)\}$ can be seen as an unnormalised probability distribution over $\mathcal{D}$.[8] A proper probability distribution can be obtained by normalisation $\bar{p_\Lambda}(\mathbf{d}) = \frac{\exp\{f(\mathbf{d},\Lambda)\}}{\sum_{\mathbf{d}' \in \mathcal{D}} \exp\{f(\mathbf{d}',\Lambda)\}}$. The m-dimensional vector of real-valued weights Λ assigns a relative importance to different aspects of the derivations captured by m feature functions $\mathbf{H}(\mathbf{d}) = \langle H_1(\mathbf{d}), \ldots, H_m(\mathbf{d})\rangle \in \mathbb{R}^m$.

If feature functions are assumed to decompose additively over the steps in a derivation, the weight of a derivation will be the sum of the weights of its steps. That is, if $H_i(\mathbf{d}) = \sum_{e \in \mathbf{d}} h_i(e)$, where h_i is a feature function that assesses steps independently (a.k.a. local feature function) and $\mathbf{d} = \langle e_1, e_2, \ldots, e_l\rangle$ is a sequence of l steps, then each step is assigned a weight equivalent to $w(e) = \Lambda \cdot \mathbf{h}(e)$, where $\mathbf{h}(e) = \langle h_1(e), h_2(e), \ldots, h_m(e)\rangle$ is a vector of local features. Interpreting weights like that is convenient in that we can make use of weighted sets and semirings to represent and operate over these quantities efficiently. However, imposing this structural independence between steps either restricts feature functions to performing very local assessments (e.g. assessing an isolated context-free production), or

[8]This formulation is known as a log-linear model (Och and Ney, 2002; Och, 2003), an instance of the maximum entropy framework (Berger et al., 1996b). Note that the *best derivation rule* shown in Equation 2.5 is a direct application of this model due to the monotonicity of the exponential function.

it increases the cost of representing the space of weighted derivations in that every step must encode all nonlocal dependencies required for assessment (e.g. assessing a prefix string in the yield of a partial derivation).

The set $\mathcal{D}$ is typically finite,[9] however, it contains a very large number of structures — exponential with the size of $\mathbf{x}$ — making exhaustive enumeration of these solutions prohibitively slow. Only in very restricted cases combinatorial optimisation techniques are directly applicable, thus it is common to resort to heuristic techniques in order to find an approximation to $\mathbf{d}^*$. In some cases the decoding problem is NP-complete — this is the case for a common model based on a finite-state representation (known as phrase-based model). Sometimes there exists an algorithm which terminates in polynomial time, but this algorithm is too slow to be practicable — this is the case for a type of model based on a context-free representation (known as hierarchical phrase-based model).

Finding the optimum derivation under the unnormalised distribution defined by the model is only one of several possible decision rules. In this thesis two classes of decision rules are addressed, the traditional decoding rule, an **optimisation** problem, and decision rules based on **sampling** — the problem of drawing samples from the normalised distribution $\bar{p_\Lambda}(\mathbf{d})$. An example of the latter is Minimum Bayes Risk (MBR) decoding, where rather than selecting the translation that maximises the model, one selects, out of a representative sample of the distribution, the translation that minimises a loss function. Decision rules and loss functions will be discussed in Section 3.2. Moreover, in this thesis strong guarantees are sought, such as reaching optimality, and drawing independent samples from the true distribution. In order to achieve that we will investigate in detail what makes phrase-based

[9]This thesis only addresses models of translational equivalence that produce strings of finite length, however, there are formulations which model word insertion producing sentences of unbounded length (Brown et al., 1993).

and hierarchical SMT hard problems — which is mostly related to characteristics of the set $\mathcal{D}$ and the parameterisation $p_\Lambda(\mathbf{d}) = \exp\{f(\mathbf{d}, \Lambda)\}$.

2.4.1 Models of translational equivalence

In **phrase-based** SMT (Koehn et al., 2003), building blocks are aligned pairs of phrases, or *biphrases*, where a phrase is simply a contiguous sequence of words, rather than a linguistically motivated unit.[10] These bilingual mappings can be represented using a cascade of wFSTs (Kumar et al., 2006), a generalisation of wFSAs for multiple vocabularies (see Sections 2.2.1 and 2.2.2). Moreover wFSTs are *closed under composition*, which means that a cascade of finite-state transducers is in fact equivalent to a wFST (Hopcroft and Ullman, 1969).

A derivation in phrase-based models is therefore represented by a sequence of transitions in a wFST, these transitions cover the input text in arbitrary order generating the output from left to right. An important constraint in phrase-based SMT is that each source word must be translated exactly once. Using a wFST, encoding information about the subsets of the input covered by each hypothesis (a partial derivation), therefore preventing overlaps, requires 2^I states, where I is the length of the input. This *non-overlapping* constraint is the source of NP-completeness of phrase-based models (Zaslavskiy et al., 2009) — which is also the case in word-based models (Knight, 1999). To lower the space complexity of such models, reordering may be restricted to happen within a limited window given by some notion of distortion limit. In fact there are several strategies for limiting reordering and they often result in non-equivalent spaces of solutions being represented. Lopez (2009) formalises the most popular reordering strategies and discusses their time and space complexities. Typically, complexity is lowered to a

[10] Simard et al. (2005) and Galley and Manning (2010) successfully model discontiguous, non-hierarchical, phrases.

polynomial function of I and an exponential function of the distortion limit (which is set to a small constant, e.g. 4-6).

In **hierarchical phrase-based**, or simply, hierarchical SMT (Chiang, 2005), building blocks are synchronous context-free rewrite rules. That is, rules of the kind $\alpha \to \langle \beta, \gamma, \sim \rangle$, where α is a variable, β is a sequence of variables and input words, β is a sequence of variables and output words, and $\sim$ is an alignment function defining a one-to-one correspondence between the variables in β and γ. We call α the rule's left-hand side and $\langle \beta, \gamma \rangle$ the rule's right-hand side, being β the input and γ the output. These bilingual mappings can be compactly represented using wSCFGs, a generalisation of wCFGs for multiple vocabularies (see Sections 2.2.3 and 2.2.4).[11] Moreover, hierarchical models of the kind introduced by Chiang (2005) are typically restricted to handling up to two nonterminals on the right-hand side of rules, being weakly equivalent to ITGs (Wu, 1997).

A derivation in hierarchical models is therefore represented as a sequence of rule applications in a wSCFG. These rules replace one variable at a time, synchronously on the source (input) and on the target (output) side. A derivation defines a parse tree over the input text which maps onto an isomorphic tree on the target side, which in turn projects onto a translation. While phrase-based reordering is arbitrary — one allows any or at least a subset of the possible permutations of the input — in the hierarchical case, the synchronous grammar encodes a polynomial number of reordering operators learnt from the parallel data.

[11]Other formalism include, but are not limited to: Tree-Adjoining Grammars (TAGs) (Shieber and Schabes, 1990), Push-Down Automata (PDAs) (Iglesias et al., 2011) and Tree-Substitution Grammars (TSGs) (Schabes, 1990).

2.4.2 Parameterisation

A typical parameterisation of a **phrase-based** model includes features such as (1) forward and backward phrase-translation probabilities, estimated as the relative frequency that a certain source phrase is translated into a certain target phrase and vice-versa in a large bilingual corpus; (2) some sort of lexical smoothing that estimates how well words align to each other within a phrase pair; (3) a phrase- and a word-count, which together control whether the model prefers fewer and longer phrases to several short phrases; (4) a distortion cost, which captures to which degree the input text is translated out of order; and (5) an n-gram language model feature that binds sequence of target words together rewarding/penalising for fluency. Observe that while (1) to (3) are local and only depend on individual choices of biphrases, (4) depends on pairs of biphrases adjacent in a derivation, and (5) depends on a finite history of target words yielded. Equation 2.6 illustrates this parameterisation: ψ is the linear combination of features over sequences of target words in the yield $\mathbf{y} = \text{yield}(\mathbf{d})$, ϕ is the linear combination of features that decompose over phrase pairs directly, and δ is the linear combination of features that require pairs of adjacent biphrases, such as the distortion cost. Other features typically fall in one of these three categories.

$$f(\mathbf{d}) = \psi(\text{yield}(\mathbf{d})) + \sum_{i=1}^{l} \phi(e_i) + \sum_{i=1}^{l-1} \delta(e_i, e_{i-1}) \tag{2.6}$$

A typical parameterisation of a **hierarchical** model includes features such as (1) forward and backward rule-translation probabilities; (2) lexical smoothing; (3) phrase and word counts, which together control whether the model prefers longer or shorter derivations; (4) a "glue rule" count, which controls the model's preference for hierarchical over serial combination of phrases; and (5) a language model feature over target n-grams. Again, most features, (1) to (4), are local and only depend on individual choices of synchronous rules, while (5) depends on a history of target words yielded. Note that in this formulation there are no mechanisms other than the grammar itself to model reordering phenomena, however the formulation could be extended to include soft constraints on reordering (Chiang et al., 2008). In Equation 2.7, ψ is the linear combination of the features over target n-grams in $\mathbf{y} = \text{yield}(\mathbf{d})$, and ϕ is the linear combination of features that decompose over rules directly.

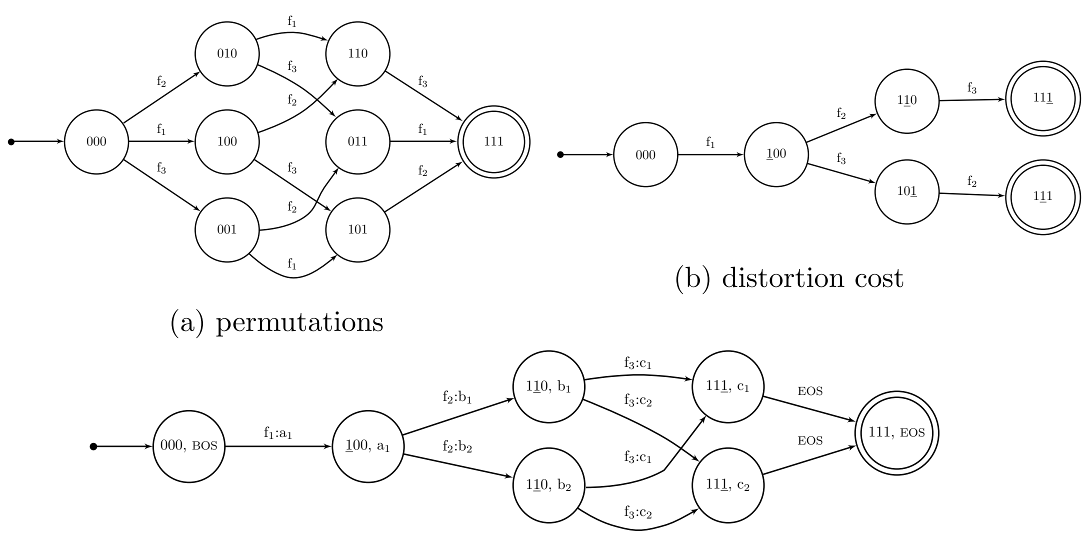
$$f(\mathbf{d}) = \psi(\text{yield}(\mathbf{d})) + \sum_{i=1}^{l} \phi(e_i) \tag{2.7}$$

With both phrase-based and hierarchical SMT, a target LM is necessary to rescore derivations. An n-gram language model defines a distribution over strings of the target language under a nth order Markov independence assumption. That is, the probability of a word given a context history is approximated to that of the given word preceded by a shortened context containing only $n-1$ words. Due to this Markov assumption, an n-gram LM can be represented by a wFSA where states represent context histories. Equation 2.8 shows the form of the LM distribution, where $\mathbf{y} = \langle y_1, \ldots, y_J \rangle$ represents an output string. This distribution is estimated from normally large monolingual corpora.

$$p_{lm}(\mathbf{y}) \equiv \prod_{i}^{J} p_{lm}(y_i | y_{i-n+1} \ldots y_{i-1}) \tag{2.8}$$

Nonlocal parameterisation of the kind of the language model component adds to the complexity of the space of weighted derivations. Recall that the linear model discussed before requires features to decompose additively over the steps in a derivation. Therefore to incorporate n-gram features, each step e must incorporate sufficient information about nonlocal dependencies.[12] In a wFSA this means that each state must remember the $n-1$

[12]In phrase-based SMT, a *step* is a transition in a wFST. Similarly, in hierarchical



(a) permutations

(b) distortion cost

(c) 2-gram language model

Figure 2.2: Example of the space of solutions of a phrase-based model

words in the suffix of the subderivations it groups. In a wCFG this means that nonterminals must encode information about $(n-1)$ words in the prefix and $(n-1)$ words in the suffix of the target strings they span.

Figures 2.2b and 2.2c illustrate the increase in the cost of representation from adding feature functions that depend on the interaction between phrases. The former shows part of the search space in Figure 2.2a expanded as to memorise at each state the last source position translated, which is necessary to compute a distortion cost. The latter illustrates the additional cost from adding a 2-gram language model component, which requires that each state memorises the last target word emitted.

Figure 2.3a illustrates the space of solutions of a hierarchical model without an LM component, the indices of the variables relate to positions between words in the input sentence "`o discurso do presidente`" (*the discourse of the president*), ranging in the interval $[0,4]$. A variable on the input

SMT, a step is a rule in a wSCFG. Due to a connection to hyperedges in hypergraphs (elaborated in Section 2.3), a step is also referred to as an *edge*.

LHS		RHS$_i$	RHS$_o$
${}_0S_4$	$\rightarrow$	<s> ${}_0X_4$ </s>	$\boxed{1}$
${}_0X_4$	$\rightarrow$	o ${}_1X_2$ do ${}_3X_4$	the $\boxed{2}$'s $\boxed{1}$
${}_0X_4$	$\rightarrow$	o ${}_1X_2$ do ${}_3X_4$	the $\boxed{1}$ of the $\boxed{2}$
${}_1X_2$	$\rightarrow$	discurso	speech
${}_1X_2$	$\rightarrow$	discurso	discourse
${}_3X_4$	$\rightarrow$	presidente	president

(a) Bilingual forest (without LM)

LHS		RHS$_o$
${}_0S_4$	$\rightarrow$	<s> ${}_0[X, \text{t} \star \text{s}]_4$ </s>
${}_0S_4$	$\rightarrow$	<s> ${}_0[X, \text{t} \star \text{d}]_4$ </s>
${}_0S_4$	$\rightarrow$	<s> ${}_0[X, \text{t} \star \text{p}]_4$ </s>
${}_0[X, \text{t} \star \text{s}]_4$	$\rightarrow$	the ${}_3[X, \text{p}]_4$'s ${}_1[X, \text{s}]_2$
${}_0[X, \text{t} \star \text{d}]_4$	$\rightarrow$	the ${}_3[X, \text{p}]_4$'s ${}_1[X, \text{d}]_2$
${}_0[X, \text{t} \star \text{p}]_4$	$\rightarrow$	the ${}_1[X, \text{s}]_2$ of the ${}_3[X, \text{p}]_4$
${}_0[X, \text{t} \star \text{p}]_4$	$\rightarrow$	the ${}_1[X, \text{d}]_2$ of the ${}_3[X, \text{p}]_4$
${}_1[X, \text{s}]_2$	$\rightarrow$	speech
${}_1[X, \text{d}]_2$	$\rightarrow$	discourse
${}_3[X, \text{p}]_4$	$\rightarrow$	president

(b) Target forest + 2-gram LM

Figure 2.3: Example of the space of solutions of a hierarchical model

right-hand side (RHS$_i$) of the form ${}_iX_j$ represents a nonterminal category X spanning from position i to position j in the source. A boxed index on the output right-hand side (RHS$_o$) of the form $\boxed{k}$ is a slot associated with the kth variable on the input right-hand side. That is, a rule application that replaces the input variable jointly rewrites the boxed index as well. Figure 2.3b illustrates the increase in the cost of representation from adding a 2-gram LM feature. Nonterminals on the target grammar are annotated with target strings. They are represented as ${}_i[X, y_1 \star y_m]_j$, signifying that X covers $i \ldots j$ in the source, and it yields the target string $y_1 \star y_m$, where the infix has been elided. The prefix and the suffix are made explicit, so that the LM component can be computed for rules combining nonterminals.

The space of weighted derivations that represents the goal distribution is equivalent to the intersection between the space of derivations weighted only locally, that is a wFSA in phrase-based models or a wCFG in hierarchical models, and the automaton that represents the language model (Chiang, 2005; Kumar et al., 2006; Dyer et al., 2008). Even when the space of derivations is polynomial before incorporating the LM component — which in phrase-based models is only the case under the assumption of a distortion limit — its intersection with a full LM is too slow to be practicable (see Section 3.3). In practice rather than instantiating an n-gram LM as a wFSA and performing the explicit intersection, one resorts to approximation techniques such as beam-search (Koehn et al., 2003) and cube-pruning (Chiang, 2007). These techniques build explicitly only a portion of the full space of derivations while searching for a high scoring derivation in an A* fashion (Hart et al., 1968). However, they do not rely on admissible heuristics and optimality is not guaranteed. Moreover, because they prune the distribution in arbitrary ways, sampling cannot be done in an exact sense.

2.4.3 Decoding

In this section we revisit two algorithms for approximate decoding that are particularly popular, namely, *beam-search* and *cube-pruning*. The former concerns finite-state models of the kind introduced by Koehn et al. (2003). The latter is used with context-free models of the kind introduced by Chiang (2005). They trade time efficiency for search errors by relying on fast-to-compute heuristics and pruning.

Beam-search (Koehn et al., 2003) is a search strategy that builds a directed acyclic graph representing a subset of the complete space of derivations implicitly defined by the model. A path in this *lattice* represents a derivation. A partial derivation is called a *hypothesis*. The algorithm proceeds by translating arbitrary segments of the input using the translation options available in a collection of biphrases (the phrase table). Hypotheses are expanded by addition of one such phrase pair at a time. The output string is built from left to right, in the target language order, and the input string is covered in arbitrary order. A single hypothesis can be expanded in many different ways depending on the choice of input phrase to cover next and the translation it yields. A hypothesis that translates each and every word of the input exactly once is a (complete) derivation.

A dynamic program that implements this strategy uses states to represent hypotheses. Their signature contains: (1) a bit vector that stores coverage information used to prevent biphrases from overlapping in the source, that is, it prevents source words from being translated more than once; (2) the last source word of the most recently chosen biphrase, necessary to compute a distortion cost; and (3) the last $n-1$ target words in the hypothesis, necessary to compute an n-gram LM feature. States are organised in stacks (priority queues) according to how many source positions their underlying hypotheses cover. A transition e represents the selection of a biphrase, it has a tail $t[e]$ (its origin state) and a head $h[e]$ (its destination state), and it covers a source span $f[e] \ldots l[e]$ producing target words $o[e]$. Consider suffix$(\cdot)$ a function that takes a state and returns the last $n-1$ target words yielded. Similarly, let last$(\cdot)$ be a function that takes a state and returns the last position covered in the source. A transition e is weighted by $\psi(\text{suffix}(t[e]) + o[e]) + \phi(e) + \delta(\text{last}(t[e]), f[e])$, where $\text{suffix}(t[e]) + o[e]$ is the string resulting of the concatenation of the suffix stored in the origin $t[e]$ and the output phrase associated with e.

The algorithm starts with a null hypothesis and follows by exhaustive expansion. However, stacks have a maximum size. If this limit is to be exceeded, the worst hypotheses are pruned. The maximum stack size realises a form of pruning called *histogram pruning*. Besides, hypotheses that are too far from the best hypothesis in the stack are discarded. How far a hypothesis can be is controlled by a parameter called the *beam width* which realises a form of pruning called *threshold pruning*. A heuristic “future cost” estimates the cost of finishing the translation from any given state in the lattice. It is cheaply computed by combining the best translation options scored disregarding reordering and complex interactions between phrase segmentation and the language model. The future cost is not an admissible heuristic, there is no guarantee that it represents the cheapest possible way of completing a partial derivation. Therefore, beam-search gives no guarantee of optimality, not even within its own pruned search space. Beam-search's popularity is due to its empirical efficacy and efficiency, and robust open-source implementations (Koehn et al., 2007; Dyer et al., 2010).[13]

Cube-pruning (Chiang, 2007; Huang and Chiang, 2007) uses a dynamic program based on CKY parsing (Kasami, 1965; Younger, 1966; Cocke and Schwartz, 1969) to parse the input sentence bottom-up while generating translations. The algorithm builds an acyclic directed hypergraph representing a subset of the complete space of derivations implicitly defined by the model. It proceeds covering contiguous spans in the source by matching them against rewrite rules in a collection of synchronous context-free rules (the rule table). These contiguous spans do not contain only words (terminals, in grammar parlance), but also variables (nonterminals) that represent source spans already covered. Unlike the phrase-based case, hypotheses are not expanded by concatenation of biphrases left-to-right in target language order. Instead, hypotheses are expanded by grouping adjacent spans under a single nonterminal. This process goes on forming larger spans until a single nonterminal covers the entire source sentence.

A dynamic program that implements this search procedure uses states (items, in parsing parlance) to represent hypothesis. An item, denoted by $[X, i, j, \gamma_1 \star \gamma_2]$, contains: (1) a contiguous source span $i \ldots j$; (2) the nonterminal X covering that span; and (3) information used for computation of LM

[13]For a sentence of length I, the complete space of derivations would require $O(2^I I^{2+n})$ states: the bit vector contributes with 2^I, the distortion feature contributes with I, the segmentation of the source contributes with I^2, and the n-gram LM feature contributes with I^{n-1}. With pre-determined limits on distortion, phrase length, size of the vocabulary of target phrases, stack size (histogram pruning) and beam width (threshold pruning), decoding can be done in time linear with the input length I (Koehn et al., 2003).

weights, namely, a target yield where all but $n-1$ words in the prefix (γ_1) and $n-1$ words in the suffix (γ_2) are elided. In this program, an edge e connects a sequence of states (the edge's tail $t[e]$) to a single state (the edge's head $h[e]$) and is associated with a synchronous context-free rule $\alpha \rightarrow \langle \beta, \gamma, \sim \rangle$. The edge's weight is given by $\psi(\text{yield}(h[e])) + \phi(\alpha \rightarrow \langle \beta, \gamma, \sim \rangle)$, where ψ scores only the complete n-grams in yield($h[e]$), and ϕ scores the synchronous rule locally. The edge's tail is directly associated with the input right-hand side β. It combines items spanning adjacent sequences in the source from left to right. The synchronous rule tells how these spans are reordered and translated yielding a target string.

The items are organised in a chart with cells (priority queues). A cell $[X, i, j]$ groups all items covering the input span $i \dots j$ with X. Cells play a similar role to that of stacks in beam-search. To limit parsing complexity, rules are limited to having up to two nonterminals on their right-hand side. Therefore, in the general case, a rule has the form

$$X \rightarrow \langle x_q^{i-1} X x_{j+1}^{k-1} X x_{l+1}^{r}, \gamma_1 X \gamma_2 X \gamma_3, \sim \rangle$$

where x_a^b is a sequence of input terminals, and γ_i is a (possibly empty) sequence of output terminals. Observe that this rule combines items from cells $[X, i, j]$ and $[X, k, l]$ producing items in cell $[X, q, r]$. The number of items generated by this rule is proportional to $O(|\Delta|^{4(n-1)})$, where $|\Delta|$ is the size of the target vocabulary (typically a function of the input length I). The exponent is due to the interactions between complete n-grams on the target right-hand side ($\gamma_1 X$, $X\gamma_2$, $\gamma_2 X$ and $X\gamma_3$).

The time complexity without any pruning is $O(I^3|\Delta|^{4(n-1)})$, where the number of possible segmentations of the input is proportional to I^3. This algorithm is too slow to be practicable even for short sentences using a 2-gram language model. To speed up the algorithm, the number of items in a cell is limited by a predetermined constant, and hypotheses are discarded if they score worse than the best item in the cell by a predetermined factor.

These parameters are equivalent to the stack size and the beam width in beam-search. For the purpose of pruning, an inadmissible heuristic accounts for partial LM information attempting to establish more meaningful comparison of items within a cell. Finally, items are not exhaustively combined by the application of a rule, instead they are combined until the cell is filled up or an item violating the beam width is produced. This enumeration is only approximately monotonic, thus items are not always generated best-first (Huang and Chiang, 2007).

2.5 An abstract view of SMT

An abstract formulation of SMT built upon an algebra of weighted sets and relations was introduced by Dyer (2010). In his view, translation is formally described in terms of operations between weighted sets and weighted binary relations.

Let X be the weighted set that represents the input text and $\mathcal{X}$ be its identity-transducer.[14] Let $\mathcal{G}$ be the weighted binary relation that compactly represents the translational equivalences between the source and the target languages.[15] Finally, let A be the weighted set of strings of the target language, also known as the LM, and $\mathcal{A}$ its identity-transducer. A formal model of translation can be seen as the composition of these three objects. The space of weighted translations is the output projection of the composition of these three weighted binary relations, as shown in Equation 2.9.[16]

[14]Typically the input is a single sentence (which can be represented as an unweighted singleton), however such generalisation makes it natural to move on to weighted word-lattices and other compact representations of ambiguity (Dyer, 2010).

[15]This object represents the bilingual mappings (including reordering) of strings. In different formalism it is represented by different computational tools, such as weighted finite-state automata, weighted context-free grammars, weighted tree-substitution grammars, etc (Lopez, 2008).

[16]Traditionally the letters f and e are used to denote source and target variables, re-

$$(\mathcal{X} \circ \mathcal{G} \circ \mathcal{A}) \downarrow \tag{2.9}$$

In phrase-based SMT, the translation model is equivalent to a wFST (Kumar et al., 2006). In hierarchical phrase-based SMT, the translation model is equivalent to a wSCFG (Chiang, 2005). These two objects can be encoded as hypergraphs (Gallo et al., 1993), a generalisation of graphs for edges with multiple origin nodes, such as it is the case with context-free grammars.

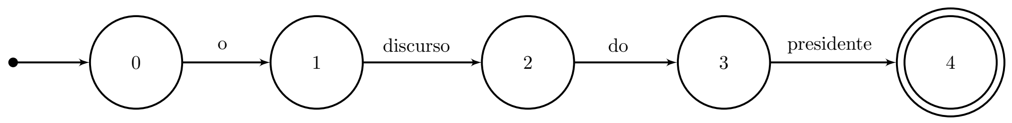
In this work the input object will always be an unweighted singleton $\mathbf{x}$. Therefore, let $G(\mathbf{x}) \equiv (\mathcal{X} \circ \mathcal{G}) \downarrow$ represent the translation hypergraph obtained by output-projection of the composition between the input and the model of translational equivalences, that is, the weighted set of translations before incorporating the language model. It is worth highlight that the methods described here generalise straightforwardly to weighted sets of input strings.

Figure 2.4a illustrates $\mathbf{x}$, an input sentence encoded as an FSA. Figure 2.4b illustrates $\mathcal{G}$ (weights are omitted in the illustration), a wSCFG expressing the known translational equivalences. The composition $\mathcal{X} \circ \mathcal{G}$ (Figure 2.4c) instantiates the grammar producing an acyclic hypergraph. Finally, the projection (Figure 2.4d) is the set of all possible tree-structured translations of the input.

To complete the translation process one needs to re-weight the possible strings by incorporating the language model, which is achieved with the weighted intersection $G(\mathbf{x}) \cap A$. Finally, a decision rule (see Section 3.2) will select a translation or a set of translations from the resulting object.

This abstract formulation helps understand the characteristics of the space of weighted derivations defined by the model. However, due to the magnitude of these objects one typically resorts to different pruning strate-

spectively. In this work we use x and y to avoid confusion, particularly with edges which are denoted by e.



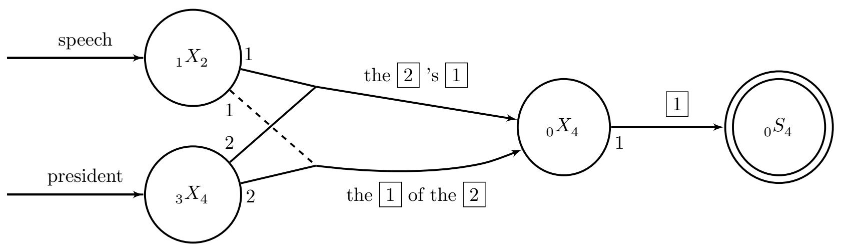
(a) Input X

LHS	RHS$_i$	RHS$_o$
S →	X	$\boxed{1}$
X →	o X do X	the $\boxed{2}$'s $\boxed{1}$
X →	o X do X	the $\boxed{1}$ of the $\boxed{2}$
X →	discurso	speech
X →	presidente	president

(b) Translation model $\mathcal{G}$

LHS	RHS$_i$	RHS$_o$
${}_0S_4$ →	${}_0X_4$	$\boxed{1}$
${}_0X_4$ →	o ${}_1X_2$ do ${}_3X_4$	the $\boxed{2}$'s $\boxed{1}$
${}_0X_4$ →	o ${}_1X_2$ do ${}_3X_4$	the $\boxed{1}$ of the $\boxed{2}$
${}_1X_2$ →	discurso	speech
${}_3X_4$ →	presidente	president

(c) Instantiated grammar $\mathcal{X} \circ \mathcal{G}$



(d) Translation hypergraph $G(\mathbf{x}) = (\mathcal{X} \circ \mathcal{G}) \downarrow$

Figure 2.4: An example of a translation hypergraph

gies which can affect each object individually, or the composition as a whole. Nevertheless, this abstract formulation gives us a view of the combinatorial discrete space over which inference is to be done and allows to formalise it with standard computational tools and algorithms.

2.5.1 Complexity

In SMT, the space of all possible mappings between bitexts is typically finite and can be represented by finite-state or context-free transducers.

In the general case, in which words may be translated in arbitrary order — any of its possible permutations — a finite-state model is NP-complete. This can be proven by connection to the Travelling Salesman Problem (TSP)

(Knight, 1999; Zaslavskiy et al., 2009). On the other hand, a context-free model takes polynomial time and space. The grammar encodes a polynomial number of reordering operators and the intersection with a finite-state language model is also polynomial. Nevertheless, exact intersection algorithms are too slow to be practical.

Another complication is that the bilingual alignment, that is, the way translational equivalences decompose into finite-state or context-free building blocks, adds to the ambiguity of the model. In other words, these models describe distributions over derivations (where the word derivation is used to refer to a solution in either a finite-state or a context-free model), while, ultimately, one wishes to reason about target strings. Reasoning about strings requires $\oplus$-summing away the ambiguous structures, that is, summing over all of the structures that project onto the same output string. Sima'an (1996, 2002) shows that, in the general case, reasoning about strings under such ambiguous representations is NP-complete for either types of models.

Therefore, it is typical in SMT, and other NLP applications, to reason about derivations instead of string themselves. In probabilistic reasoning, that relaxation implies further partitioning the probability mass, and there will be no guarantees that the best derivation complies with the best string. In this thesis we reason exactly over derivations both in optimisation and in sampling. Moreover, an exact sampler over derivations is also an exact sampler over strings, which in certain cases may lead to more theoretically sound approximations to the problem of finding the best string.

Chapter 3

Inference

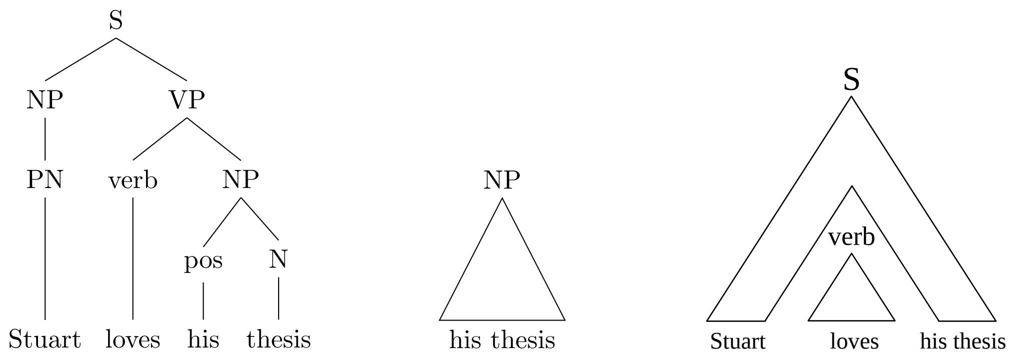
This chapter is divided in three main parts. The first part (Section 3.1) is a revision of Inside-Outside, an important inference algorithm generalised to semirings (see Section 2.1). It also shows that for tractable hypergraphs, sampling and optimisation can be done with a simple dynamic program that relies on the computation of inside weights. Section 3.2 formalises decision rules and surveys related work on their use in SMT. Finally, Section 3.3 surveys previous work on inference in SMT which is related to the work presented in this thesis.

3.1 Dynamic programming algorithms

This section revisits the Inside-Outside algorithm and its application, particularly of the Inside part, to sampling and optimisation from hypergraphs.

3.1.1 Inside-Outside

In probabilistic parsing, the *inside probability* is the probability that a certain sequence of terminal symbols (a phrase) is spanned by a certain nonterminal (a phrase category) in no specific context. For an intuitive illustration, consider the parse tree shown in Figure 3.1a adapted from (Goodman, 1998). The probability that the category NP spans the phrase "his thesis" is the quantity *inside*("his thesis", NP). It abstracts away the internal structure of the phrase, as shown in Figure 3.1b. The *outside probability* can be thought of as the probability of everything surrounding a certain phrase given an



(a) Parse tree (b) Inside (c) Inside-Outside

Figure 3.1: Illustration of inside and outside probabilities

input sentence. In the example, the probability that "loves" is tagged as a verb in the sentence "Stuart loves his thesis" is the quantity *outside*("Stuart $\boxed{verb}$ his thesis").

Baker (1979) introduced the Inside-Outside algorithm, a generalisation of Forward-Backward (Chang and Hancock, 1966; Baum et al., 1970) that operates with wCFGs, rather than wFSAs. Goodman (1998, 1999) generalised Inside-Outside to semirings. We revisit this algorithm here for an acyclic hypergraph $G = \langle V, \langle E, w \rangle \rangle$ and a commutative semiring K.[1] Moreover G represents a finite-state automaton or a context-free grammar, and it has a single goal node v_g, also denoted by `root`, whose weight is $\bar{1} \in \mathbb{K}$. The absence of cycles in G means that its nodes can be topologically sorted, that is, there is a partial ordering between nodes in V, such that if a node v depends on a node u, u is processed before v.

Algorithm 1 illustrates topological sorting and runs in linear time with the size $|G| = |V| + |E|$ of the hypergraph. One starts with a set S of nodes with no incoming edges (line 2), that is, nodes that do not depend on any other node, and a mapping between nodes and their direct dependencies (line 3), where a node depends (directly) on the tail nodes of its incoming

[1] Goodman (1998) discusses the computation of inside and outside weights from cyclic hypergraphs as well as using non-commutative semirings.

Algorithm 1 Topological sort

```
1: function TopSort($G = \langle V, \langle E, w \rangle \rangle$)
2:     $S = \{q \in V : BS(q) = \emptyset\}$                          ▷ states with no dependencies
3:     $D = \{v \mapsto \{u : \exists e \in BS(v) \wedge u \in t[e]\} : v \in V\}$      ▷ dependencies
4:     $L = \langle\rangle$                                          ▷ top-sorted nodes
5:     while $S \neq \emptyset$ do
6:         $q \leftarrow \mathrm{pop}(S)$                       ▷ remove and return a node from $S$
7:         $L \leftarrow L + \langle q \rangle$                              ▷ append $q$ to $L$
8:         for $e$ in $FS(q)$ do                          ▷ outgoing edges from $q$
9:             $p \leftarrow h[e]$                                 ▷ parent of $q$ in $e$
10:            $D(p) \leftarrow D(p) \setminus \{q\}$                    ▷ remove $q$ from $D(p)$
11:            if $D(p) == \emptyset$ then             ▷ $p$'s dependencies have been sorted
12:                $S \leftarrow S \cup \{p\}$
13:            end if
14:        end for
15:    end while
16:    return $L$
17: end function
```

edges. A node in S is such that its dependencies have already been processed. Therefore, nodes in S are popped one at a time, added to the top-sorted list and removed from the dependencies of their parents (head nodes of the outgoing edges). When all of a node's dependencies have been sorted, this node can be added to S (line 12). The algorithm terminates when S is empty.

The inside weight of a node q is the total weight under that node in G.[2] It is given by Equation 3.1, where $\mathbf{d}$ is a derivation and e is an edge in $\mathbf{d}$. Algorithm 2 illustrates a dynamic program to compute inside weights which runs in time $O(|G|)$.

$$\mathsf{I}[q] = \bigoplus_{\mathbf{d} \in G} \bigotimes_{e \in \mathbf{d}} w(e) \tag{3.1}$$

In the sum-times semiring, $\mathsf{I}[q]$ can be seen as the total probability mass under q. The quantity $\mathsf{I}[v_g]$ is the sum of weights of all derivations in the hypergraph. That is, the Inside algorithm provides an efficient way of com-

[2]Because G's nodes have a partial ordering, it is possible to imagine G as a tree structure, in which nodes with no dependencies are at the bottom (leaves) and the goal node is at the top (root). The total weight under q refers to the sum of weights of all subderivations (top-down) starting from q.

Algorithm 2 Inside weights

```
function INSIDE($G = \langle V, \langle E, w \rangle\rangle$)
    for $q$ in TOPSORT($G$) do                                  ▷ visit nodes bottom-up
        incoming $\leftarrow B(q)$
        if incoming $== \emptyset$ then
            $\mathsf{I}[q] \leftarrow \bar{1}$                                  ▷ leaves
        else
            $\mathsf{I}[q] \leftarrow \bar{0}$
            for $e$ in incoming do          ▷ total inside weight of an incoming edge
                $k \leftarrow w(e)$                      ▷ including the edge's own weight
                for $r$ in $t[e]$ do
                    $k \leftarrow k \otimes \mathsf{I}[r]$
                end for
                $\mathsf{I}[q] \leftarrow \mathsf{I}[q] \oplus k$                    ▷ accumulate for each edge
            end for
        end if
    end for
    return $\mathsf{I}$
end function
```

puting the *partition function* used to normalise the distribution defined by the hypergraph, which enables, for instance, probabilistic sampling. In the max-times semiring, $\mathsf{I}[q]$ represents the maximum weight under q. Therefore, the quantity $\mathsf{I}[v_g]$ is the weight of the best derivation in the hypergraph, which can be used to perform maximisation.

The outside weight of a node q is the total weight surrounding q, but excluding that node. In probabilistic parsing it is associated with the total probability around a phrase. Algorithm 3 illustrates a dynamic program to compute outside weights which runs in time $O(|G|^2)$. The outside weight is such that $\mathsf{I}[q] \times \mathsf{O}[q]$ is the marginal weight of q in G. In a probability semiring, the marginal is the expectation under the model of sampling q from G. Algorithm 4 shows how the marginal weights of nodes and edges can be computed in linear time $O(|G|)$ using pre-computed Inside-Outside weights. Edge and node marginals can be used to identify parts of the hypergraph that are more likely than others. Sixtus and Ortmanns (1999) propose a pruning

Algorithm 3 Outside weights

```
1: function OUTSIDE($G = \langle V, \langle E, w\rangle\rangle, \mathsf{I}$)
2:     for $q \in V$ do                                        ▷ these are all nodes
3:         $\mathsf{O}[q] \leftarrow \bar{0}$
4:     end for
5:     $\mathsf{O}[\text{root}] \leftarrow \bar{1}$            ▷ this is the goal node — root symbol
6:     for $q$ in REVERSE(TOPSORT($G$)) do                     ▷ visit nodes top-down
7:         for $e \in BS(q)$ do                                ▷ $q$'s incoming edges
8:             for $r$ in $t[e]$ do                            ▷ children of $q$ in $e$
9:                 $k \leftarrow w(e) \otimes \mathsf{O}[q]$
10:                for $s$ in $t[e]$ do                        ▷ siblings of $r$ in $e$
11:                    if $r \neq s$ then                      ▷ thus $r$ itself must be excluded
12:                        $k \leftarrow k \otimes \mathsf{I}[s]$   ▷ incorporate inside weights surrounding $r$
13:                    end if
14:                end for
15:                $\mathsf{O}[r] \leftarrow \mathsf{O}[r] \oplus k$   ▷ accumulate it for $r$
16:            end for
17:        end for
18:    end for
19:    return $\mathsf{O}$
20: end function
```

strategy for word graphs based on marginal probabilities and call it *forward-backward pruning*. Caraballo and Charniak (1998) extend the strategy to probabilistic context-free grammars. In SMT, Graehl (2005) and Huang (2008) use edge marginals to prune translation forests. In this thesis, edge marginals are used to selectively intersect a hypergraph with an automaton expanding the hypergraph only in regions that are likely to participate in high-scoring derivations.

3.1.2 Sampling and optimisation

Sampling and optimisation can be seen as two extremes in a continuum of inference tasks in L^p spaces (Dymetman et al., 2012b), an argument that will be better established in Section 4.1. One of the main challenges of sampling from a multi-dimensional distribution is computing this distribution's partition function, that is, $\oplus$-summing over all the exponentially many derivations

Algorithm 4 Marginal weights

```
1: function NodeMarginals($G = \langle V, \langle E, w \rangle\rangle$, I, O)
2:     for $q \in V$ do                                  ▷ these are nodes
3:         $\mathsf{M}[q] \leftarrow \mathsf{I}[q] \otimes \mathsf{O}[q]$
4:     end for
5:     return M                                  ▷ marginal weight of symbols
6: end function
1: function EdgeMarginals($G = \langle V, \langle E, w \rangle\rangle$, I, O)
2:     for $e \in E$ do                                  ▷ these are edges
3:         $\mathsf{M}[e] \leftarrow w(e) \otimes \mathsf{O}[h[e]]$
4:         for $q$ in $t[e]$ do
5:             $\mathsf{M}[e] \leftarrow \mathsf{M}[e] \otimes \mathsf{I}[q]$
6:         end for
7:     end for
8:     return M                                  ▷ marginal weight of rules
9: end function
```

it represents. The inside algorithm provides an efficient way of obtaining that quantity exactly, as long as the distribution can be represented by a tractable hypergraph.

Algorithm 5 illustrates the process of sampling from the distribution associated with a hypergraph $G = \langle V, \langle E, w \rangle\rangle$, given inside weights computed from G with an appropriate semiring. As it will soon become obvious, this algorithm performs sampling in a generalised sense that also accounts for optimisation. One starts from the root of G and proceeds by iteratively selecting an edge that satisfies the criterion established by the procedure SELECT (line 6). This procedure selects an edge whose head is q out of the set of incoming edges to q, i.e. $e \in BS(q)$. The selected edge is added to the partial derivation and its tails are added to the queue of nodes to be visited. Since G is acyclic, this algorithm terminates in at most $|V|$ steps. In the end a complete derivation is produced and returned.

$$\mathcal{I}_q(e) = w(e) \otimes \left(\bigotimes_{r \in t[e]} \mathsf{I}[r] \right) \tag{3.2}$$

Algorithm 5 Generalised sampling

```
 1: function SAMPLE($G = \langle V, \langle E, w \rangle\rangle$, $\mathsf{l}$)
 2:     $\mathbf{d} \leftarrow \langle\rangle$                                ▷ an empty partial derivation
 3:     $Q \leftarrow \langle v_g \rangle$                  ▷ a queue of nodes to be visited, starting from $v_g$
 4:     while $Q$ do
 5:         $q \leftarrow \mathrm{pop}(Q)$
 6:         $e = \text{SELECT}(BS(q), \mathsf{l})$)                              ▷ select an edge
 7:         $\mathbf{d} = \mathbf{d} + \langle e \rangle$                 ▷ add the edge to the partial derivation
 8:         $Q \leftarrow Q + t[e]$          ▷ queue the nodes in the tail of the selected edge
 9:     end while
10:     return d
11: end function
```

To characterise the derivations sampled, consider the function $\mathcal{I}_q : BS(q) \subseteq E \to \mathbb{K}$ defined in Equation 3.2. This is the unnormalised distribution over incoming edges to a node q, such that, for each edge, it returns the inside weight associated to that edge, with all of its tail nodes considered, $\otimes$-times the edge's own weight. If inside weights are computed in the *sum-times* semiring, the quantity $\mathcal{I}_q(e)$ for a given edge $e \in BS(q)$ represents the total weight associated with derivations that continue top-down from q via e. If the *max-times* semiring is used instead, the quantity $\mathcal{I}_q(e)$ represents the maximum weight associated with continuing a derivation top-down from q via e.

Sampling is an instance of Algorithm 5, where we operate in the *sum-times* semiring and the procedure SELECT draws a random edge with respect to the distribution $\bar{\mathcal{I}}_q(e)$, as shown in Equation 3.3. Note that for each node q there is a finite number of edges in $BS(q)$, bounded by $|E|$ and typically much smaller than that, thus summing over $BS(q)$ to compute the normalisation term is straightforward.

$$\hat{e} \sim \bar{\mathcal{I}}_q(e) = \frac{\mathcal{I}_q(e)}{\sum_{e' \in BS(q)} \mathcal{I}_q(e')} \tag{3.3}$$

Optimisation is also an instance of Algorithm 5, where we operate in the *max-times* semiring and the procedure SELECT performs the maximisation in Equation 3.4. Finding the argmax is straightforward because $BS(q)$ is bounded and typically small.

$$e^* = \operatorname*{argmax}_{e \in BS(q)} \mathcal{I}_q(e) \tag{3.4}$$

3.2 Decision rules

In Section 2.4.3, we discussed the *decoding* process as a maximisation problem, that is, decoding required finding the derivation which maximises the objective function. This *decision rule* is not the only possible way to decide on one of the exponentially many solutions encoded in a hypergraph. In SMT, one is really interested in finding the best translation string for a given input. Selecting the best derivation is a decision motivated almost exclusively by computational complexity. Recall that the total weight of a string requires summation over all derivations that yield it. Therefore, there is no guarantee that the string yielded by the best derivation will be the best possible translation defined by the model. Other decision rules take more global information into account before committing to a solution. For instance, some decision rules rely on a summary of the distribution given by sampling. Others are based on the notion of expectation and capitalise on the entire distribution. They can also bias the selection towards some external notion of loss or quality.

If $\langle X, w \rangle$ is a weighted set, Equation 3.5 represents the *max-weight* decision rule.

$$x^* = \operatorname*{argmax}_{x \in X} w(x) \tag{3.5}$$

This decision rule has at least two specialisations, namely, *max-derivation* and *max-translation*. The former is the most popular of the decision rules (Lopez, 2008), where $\langle X, w \rangle$ is a weighted set of **derivations**, typically represented by a hypergraph, x is a derivation, and $w(x)$ its weight. The latter addresses the fact that parameterisation often splits the probability mass between the typically many structures on top of each string, that is, the fact that multiple derivations actually project onto the same translation string. A more principled decision rule would pick the one translation (**string**) that has the maximum weight with all derivations considered. Let $\langle D, u\rangle$ be a weighted set of derivations and $\langle X, w\rangle$ be the weighted set of strings yielded by derivations in D. The weight of a string $x \in X$ is such that $w(x) = \bigoplus_{d\in D(v_g,x)} u(d)$, where $D(v_g, x)$ is the set of all derivations rooted at the goal v_g whose target projection is x. The *max-translation* rule picks from all target strings in $X \subseteq \Delta^*$ the one whose total weight $\bigoplus_{d\in D(v_g,x)} u(d)$ is maximum.[3]

Note that *max-translation* requires summation over all possible structures on top of strings, an NP-complete problem (Sima'an, 1996, 2002). Note that partitioning the probability mass assigned to a string x by introduction of a latent variable, such as the structure on top of the string, does not change the total mass assigned to x. Therefore, if the sampler is exact, in that samples are independently drawn without biased correlations, this exact sampler of derivations is also an exact sampler of strings. However, the *max-translation* problem remains intractable in the general case. That is so because observing the most probable string can still require an intractable number of samples.[4]

An approximation to the *max-translation* rule is the technique of *crunching* in which one sums over derivations in an n-best list (May and Knight,

[3]In the context of a string-to-tree model, May and Knight (2006) discuss a third instance of the *max-weight* rule, which we could call *max-output-tree*. They introduce a technique to determinise a hypergaph with respect to the target trees it produces. The total weight of a target tree requires summing over all derivations that produce it (the input may be segmented and consumed in different ways yet producing the same target tree). They show that the ranking within an n-best list changes considerably by collapsing multiple derivations that project on the same target tree.

[4]We do not attempt to formally address this observation, however we remark that the intuition behind it is that the sampler error with respect to the *max-translation* criterion (i.e. sampling a solution other than the best string) remains non-polynomial with the input length. For a formal discussion with proofs refer to (Sima'an, 1996, 2002).

2006). Blunsom et al. (2008) modify cube-pruning to keep the full target string in the states of the dynamic program. As this is very expensive, they resort to aggressive pruning and can only decode short sentences (up to 10 words). Li et al. (2009) use a variational technique which considers all derivations in a hypergraph (typically obtained with beam-search or cube-pruning). Arun et al. (2009, 2010) design a Gibbs sampler, an MCMC method, for phrase-based models and find an approximation to the best translation string by marginalising over a sample of *derivations* obtained with that technique.

Deciding on a derivation that has maximum weight can be too harsh and might overlook important aspects of the distribution, such as multi-modality for instance. A more principled solution would take into account some notion of *risk* associated to choosing a particular solution as opposed to any other, and base the decision on minimising that risk.

A *loss function* defines an overall measure of *loss* incurred in choosing any of the available solutions in the weighted set $\langle X, w \rangle$. It is such that it takes a pair of translations, where one is assumed a hypothesis and the other is assumed to be a reference (or gold-standard) translation, and returns how poor an approximation the former is to the latter. If one has access to reference translations, a principled decision rule is one that minimises the loss with respect to those references. However, access to gold-standard translations is restricted to training, when parallel data is available. In the absence of parallel data, one can minimise the expected loss with respect to the normalised distribution $\bar{w}(x)$. Equation 3.6 shows this Minimum Bayes Risk (MBR) rule (Bickel and Doksum, 1977), where $\langle l(x', x) \rangle_{\bar{w}(x)} = \sum_{x \in X} w(x) l(x', x)$ is the expectation of the loss function $l : X \times X \to \mathbb{R}$ under the normalised distribution $\bar{w}(x)$. Observe that for a binary loss function, which is zero if and only if $x' = x$, minimising the expected loss is equivalent to maximising the weight $w(x)$ directly.

$$\hat{x} = \operatorname*{argmin}_{x' \in X} \langle l(x', x) \rangle_{\bar{w}(x)} \tag{3.6}$$

This form of *minimum risk* rule requires normalisation of $w(x)$. Moreover, a naïve implementation would require $O(|X|^2)$ evaluations of the loss function, which is beyond consideration given the magnitude of X. Li and Eisner (2009) introduce semirings for the computation of expectations and variances from hypergraphs and describe a generalised Inside-Outside algorithm for efficient computation of those quantities. However, to compute expectations exactly $\langle X, w \rangle$ needs to be represented by a tractable hypergraph, which is typically not the case. Moreover, the choice of loss function adds to the complexity of the representation, since sufficient statistics must be incorporated to the steps in a derivation. A lower-complexity alternative is to compute the the minimum risk rule on the basis of an unbiased sample from $\bar{w}$. However, it is often hard to obtain unbiased samples from multi-dimensional discrete distributions.

Kumar and Byrne (2004) introduces MBR decoding to SMT in the context of approximate decoding algorithms. They rely on n-best lists to apply the rule to a tractable subset of the space of possible derivations.[5] Because n-best lists are typically small (a few hundreds to a few thousands of derivations) they can work with a variety of loss functions including those based on the traditional BLEU metric.[6]

Tromble et al. (2008) extend MBR decoding to complete translation lattices (which might have been generated with some approximate method, such as beam-search) using a loss function based on a sentence-level approxima-

[5]These n-best lists are arguably poor approximations of probabilistic samples. They typically contain derivations clustered around what is heuristically believed to be the mode of the distribution (Blunsom and Osborne, 2008).

[6]Bilingual Evaluation Understudy (BLEU) is a metric for automatic evaluation of machine translation quality. It compares a translation hypothesis to one or more references based on a notion of precision of (exact) n-gram matching and a brevity penalty that prevents very short translations to score higher (Papineni et al., 2002).

tion to BLEU. Note that, while pruning the search space introduces arbitrary biases, using the whole lattice gives access to much more diverse settings of features than the n-best lists do, which results in less biased expectations. Accurate expectations enable better parameter estimation. Similarly, Kumar et al. (2009) generalise MBR decoding to complete translation hypergraphs. DeNero et al. (2009) introduce *consensus decoding*, a decision rule related to MBR that can be efficiently computed from sufficient statistics (e.g. n-gram counts) in an n-best list or in an translation hypergraph.

Arun et al. (2010) use a Gibbs sampler to obtain samples indirectly from $w(x)$ and perform minimum risk decoding. However, their sampler produces samples in a Markov chain, and it is hard to assess when, if ever, the chain has converged to a sample from the goal distribution.

3.3 Inference in SMT

This section surveys related work on approximate and exact inference (sampling and optimisation) for phrase-based and hierarchical SMT. In Section 3.3.1, we start with approximate methods to optimisation, followed by related work on approximate sampling. Section 3.3.2 surveys approaches to exact optimisation. To the best of our knowledge, this thesis presents the first approach to exact sampling in SMT (see Section 4.2).

3.3.1 Approximate inference

Dynamic Programming (DP) has always been at the core of SMT decoding, since the original IBM patents (Brown et al., 1995; Berger et al., 1996a). The first decoders were inspired by A* optimisation (Hart et al., 1968). In the dynamic program a hypothesis represents a partial derivation (the prefix of a translation). If a hypothesis covers all the words of the input, it represents a complete derivation. Hypotheses are organised in stacks (priority queues), which are limited in size and serve the purpose of grouping hypotheses according to a common trait (e.g. the number of source words covered). Organising hypotheses like that helps establish fair comparisons for the purpose of pruning: if hypotheses of different length were to compete, those covering fewer words would almost always get prioritised. In addition to that, a heuristic estimates the cost of finishing a hypothesis by translating the remaining uncovered words (rest cost, or future cost estimation). At each iteration the program decides on which hypotheses to expand (e.g. the best from each stack). The selected hypotheses are expanded by all applicable actions (e.g. covering a word/phrase with a translation option) resulting in new hypotheses that are inserted in the appropriate stacks. At this point stacks might get pruned if too large.

An admissible heuristic is such that it never underestimates the cost of finishing a translation from any given state of the search (Hart et al., 1968). That way, the dynamic program can guarantee optimality (except for eventual pruning strategies) by always expanding the cheapest amongst all (overestimated) hypotheses (a form of best-first agenda). Designing admissible heuristics is a difficult task. If the heuristic overestimates too much, a large space of solutions is explored in order to ensure optimality. If the heuristic is such that it might underestimate the rest costs (thus inadmissible), the search algorithm might finish prematurely with a sub-optimal solution. Admissible heuristics can be thought of as proxies to outside weights. Note that while the computation of outside weights requires the instantiation of the complete search space, admissible heuristics are designed to cheaply estimate those weights in a best-first strategy. Stack decoding has been extensively used in SMT decoding (Wang and Waibel, 1997; Germann et al., 2001; Och et al., 2001; Tillmann and Ney, 2003; Koehn et al., 2003).

Beam-search (Koehn et al., 2003) and cube-pruning (Chiang, 2007) are modern instances of DP-based decoding algorithms. They rely on pruning to lower complexity to a manageable polynomial function of the input length.

They employ fast-to-compute inadmissible heuristics. Hypotheses from all stacks are exhaustively expanded up to the limits established by histogram and/or threshold pruning. They also shrink the space of solutions by hypothesis recombination (Och et al., 2001), an error-safe technique which merges hypotheses that cannot be distinguished in terms of their signatures.

Watanabe et al. (2006) restrict the hierarchical phrase-based model of Chiang (2005) to target grammars in Greibach Normal Form (GNF) (Greibach, 1965). That is, synchronous rules have the form $X \rightarrow \langle \alpha, \bar{b}\beta, \sim \rangle$, where $\bar{b}$ is a sequence of terminals and β is a sequence of nonterminals (they may be empty sequences, but not both empty). Observe that only to the target side of the grammar must be in GNF. The GNF facilitates incorporation of language model scores. The framework is similar to a phrase-based model with reordering hard-constrained by the grammar. Like in the phrase-based case, the decoder proceeds generating the translation from left to right in target language order. With respect to the language model, the states in the dynamic program need to memorise only the last $n-1$ target words generated (as opposed to the two boundaries of a span as it happens in cube-pruning).

Decoding algorithms based on **finite-state operations** implemented by general purpose finite-state toolkits are also popular in phrase-based SMT. However, they typically require strong limits in reordering (Kumar and Byrne, 2003; Kumar et al., 2006). In the context of hierarchical models, Gispert et al. (2010) replace cube-pruning by standard wFST operations. First the input is parsed using CKY with hypothesis recombination, but without pruning. Then a target language word lattice is generated from each cell in the chart. Recall that a cell groups hypothesis that cover a span $i \dots j$ under nonterminal X, denoted $[X, i, j]$. Thus, each lattice represents all the translations generated by subderivations in the cell. Constructing such lattices requires processing nested items. A recursive procedure controls for efficient construction of the final lattice using union, concatenation and replacement operations with OpenFST (Allauzen et al., 2007). Pruning happens in a controlled way depending on the level of the item. They work on both alignment and translation modes. In translation, they control the degree of rule nesting in derivations by modification of the nonterminal vocabulary, a technique they call shallow-n grammars. They rewrite the ruleset of the grammar such that the maximum degree of rule nesting is n. This technique helps alleviate overgeneration, a phenomenon by which the same set of rules allows different translations, particularly due to different nested configurations leading to further reordering.

An alternative to dynamic programming is **greedy search**. In greedy search, one starts from a draft derivation, such as a word-to-word gloss translation, and proceeds by the iterative application of operators designed to transform the solution. At each step, the algorithm selects the best derivation one operation away from the current one. The success of this strategy depends on how much of the search space the operators can actually cover, however, there are no guarantees of optimality. The operators should be designed in such a way that they are not trapped in local maxima, they modify hypotheses in a non-trivial manner, and they are inexpensive. Germann et al. (2001) first introduced greedy search to decode with word-based models (Brown et al., 1990). Greedy strategies have also been proposed to phrase-based and hierarchical models (Marcu and Wong, 2002; Eisner and Tromble, 2006; Langlais et al., 2007). They typically start from a draft translation and hillclimb towards better hypotheses.

Sampling from intractable discrete distributions is a difficult task. A simple and not necessarily effective strategy is to **sample heuristically** from a truncated hypergraph produced by a DP-based decoder. This pruned hypergraph contains only a subset of all possible derivations that are compatible with the input. Moreover, pruning adds **arbitrary bias**, therefore, there is little hope that this procedure will produce samples according to the underlying model. Nevertheless, this technique might be sufficient to improve the diversity of n-best lists (with respect to the derivations they contain) in parameter estimation, a common scenario to which this technique has been applied (see below).

Blunsom and Osborne (2008) address probabilistic inference in the context of hierarchical translation, where sampling is used both for the purposes of decoding and parameter estimation. They employ cube-pruning to construct a pruned wCFG which generates a subset of all the possible derivations that would correspond to a full intersection with the target language model. Then, they sample heuristically from this wCFG through the same procedure described in Section 3.1.2. By contrast to our approach, they do not attempt to perform exact inference. However they do not only address the question of decoding, but also that of training the model, which requires, in addition to sampling, an estimate of the model's partition function. Chatterjee and Cancedda (2010) use sampling to improve parameter estimation for phrase-based models. They use beam-search to produce a truncated translation lattice from which they sample heuristically. Samples are used instead of n-best lists to perform MERT directly.

MCMC algorithms have been designed for phrase-based and hierarchical SMT. In **Gibbs sampling** (Geman and Geman, 1984), samples are obtained by iteratively re-sampling groups of well-designed variables such that (i) the sampler does not tend to be trapped locally by high correlations between conditioning and conditioned variables, and (ii) the combinatorial space of possibilities for the next step is small enough so that conditional probabilities can be computed explicitly. By contrast to our exact approach, the samples obtained by Gibbs sampling are not independent, but form a Markov chain that only converges to the target distribution in the limit, with convergence properties that are difficult to assess.

Arun et al. (2009, 2010) address sampling in phrase-based models. They use samples to decode with different decision rules, namely, *max-derivation*, *max-translation* and MBR decoding. By contrast to this thesis, they do not address the question of finding the maximum derivation directly, but do so only through finding a maximum amongst the derivations sampled so far, which in principle can be quite different. Roth et al. (2010) design a similar Gibbs sampler for phrase-based models, however, they frame translation as a conditional random field which allows overlapping biphrases. Sampling is used for decoding and for training. Parameter estimation is performed with *SampleRank* (Wick et al., 2009), a technique which encourages agreement between the model and the objective function with respect to the ranking of the solutions. Blunsom et al. (2009) design a Gibbs sampler for hierarchical models, however, in the context of inducing synchronous grammars from bilingual corpora. They work with grammars having up to three nonterminals on the right-hand side. They argue that while a ternary grammar would considerably complicate dynamic programming, it helps their sampler by giving it access to operations that can more easily reach varied permutations of a derivation's internal structures.

The operators in (Arun et al., 2009; Blunsom et al., 2009; Roth et al., 2010) are very similar to those in (Germann et al., 2001). In both greedy search and Gibbs sampling, at each step the space of derivations reachable from the current one is limited by the transformation operators, making the space of reachable solutions tractable. In greedy search, one optimises this one-step transformation. In Gibbs sampling, one samples from the distribution associated with each operation.

3.3.2 Exact inference

Under certain conditions **$\mathbf{A}^*$ style** exact optimisation is possible for word- and phrase-based models. Tillmann et al. (1997) work with word-based models and operate under the assumption that the alignment is monotone with respect to word order. Och et al. (2001) do not limit reordering, but decode exactly only short sentences using A* and admissible heuristics. They also discuss faster-to-compute almost-admissible heuristics. Kumar et al. (2006) show that phrase-based decoding can be done exactly with strong limits in reordering using standard wFST operations.

In the context of hierarchical models, the formal connections between cube-pruning and A* can be insightful. Recall that cube-pruning works by controlling item generation in synchronous parsing with an n-gram language model component. Item generation refers to the process of creating larger items by combination of previously proven items that interact in a synchronous rule. In this process, cube-pruning attempts to create items in best-first order. However, this enumeration is approximate (done with an inadmissible heuristic), thus the algorithm might prematurely stop item generation, missing out good items. Hopkins and Langmead (2009) frame item generation as a heuristic search for the top scoring combinations in a tree structure that represents the possible ways of combining items in a rule. They derive a tunable heuristic that ranges from strong (effective) inadmissible to weak (not so effective) admissible and experiment with an exact version of cube pruning which is feasible under some assumptions regarding the grammars used. The authors show an important connection between cube-pruning and A* which helps understand the idiosyncrasies of the former.

Zaslavskiy et al. (2009) formulate phrase-based decoding as an instance of the Asymmetric Generalised TSP (AGTSP) and rely on robust exact and approximate **TSP solvers** to perform fast decoding. In the AGTSP, the *salesman graph* is a weighted directed graph G and a partition of the nodes in G defining m disjoint subsets called clusters. The problem consists in finding a tour with minimal cost such that each cluster is visited exactly once. The nodes of the graph have the form (w, b) in which w represents a source word (position) in the input and b is a biphrase whose source includes w. The clusters are the subsets of nodes that share a common w. A special cluster with a single node representing the sentence boundary is also added. Nodes in different clusters are connected to be consistent with the biphrases they represent and the source words they consume. Consider two nodes (w, b) and (w', b) sharing the same biphrase b, where w and w' are consecutive in the input, the cost between these two nodes is zero, because the cost of the biphrase is taken into account once one commits to using it for the first time. If two nodes (w, b) and (w', b') are such that $w \neq w'$, $b \neq b'$ and w' is the leftmost source word in b', the cost to move from the first to the second represents the cost of committing to the biphrase b'. Features local to the biphrases, the distortion cost, and the language model component (if a 2-gram LM) can be computed directly. Whatever other pair of nodes is not connected (or connected with an infinite cost). A tour from the sentence boundary that visits each cluster exactly once and returns to the sentence boundary represents a valid derivation. To account for higher-order LMs, one strategy is to effectively multiply the number of nodes encoding an $n - 1$ context history in each node (similar to what is done in dynamic programming). However, this strategy increases considerably the salesman graph compromising efficiency beyond a 2-gram LM.

To some extent, word-based decoding had been previously formulated as an instance of the TSP by Germann et al. (2001), however, relying strictly on **Integer Linear Program (ILP)**. In the ILP formulation, an exponential number of constraints is necessary to ensure that the source words are consumed exactly once. As a consequence they work with short sentences (up to 8 words) and a 2-gram language model. Riedel and Clarke (2009) revisit that formulation and apply a cutting-plane algorithm (Dantzig et al., 1954) (again a solution inspired by TSP solvers) decoding sentences up to 30 words. In a cutting-plane algorithm, the problem is solved without (or with a tractable subset of) the exponentially many constraints that prevent input words from being translated more than once. If the solution defines a derivation in which each source word is translated exactly once, the solution is guaranteed to be optimal. Otherwise, some constraints are added and the problem is solved again. This process is repeated until the optimal solution, the one that does not violate the non-overlapping constraint, is found. This idea is also implicitly shared with methods based on Lagrangian relaxation (see below) and our own formulation based on rejection sampling (see Chapter 6).

Rush et al. (2010) introduce **dual decomposition** (a special case of **Lagrangian relaxation**) as a framework for deriving exact optimisation algorithms for NLP problems. Within this framework, Rush and Collins (2011) and Chang and Collins (2011) address exact decoding for hierarchical and phrase-based translation, respectively. Rush and Collins (2011) use a dual decomposition approach where the target wCFG component and the target language model component trade their weights so as to ensure agreement on what each component believes to be the maximum. In many cases, this technique is able to detect the actual true maximum derivation. When this is not the case, they use a finite-state-based intersection mechanism to "tighten" the first component so that some constraints not satisfied by the current solution are enforced, and iterate until the true maximum is found or a time-out is met, which results in a high proportion of true maxima being found.

Chang and Collins (2011) relax the space of solutions of a phrase-based model by allowing violations of the non-overlapping constraint. They work with a maximum distortion strategy avoiding the NP-completeness of phrase-based models. In the unconstrained lattice, dynamic programming is feasible. Soft constraints are added in the form of Lagrangian multipliers that are iteratively adjusted to penalise/reward derivations each time a source position is covered. The multipliers are easily incorporated as weights in the objective function, thus the complexity of the search space, and the search itself, remains unchanged. If a solution that complies with the non-overlapping constraint is found, it is guaranteed to be optimal. Otherwise, hard constraints are added to the lattice in the form of a coverage bit vector. That is, they explicitly add partial coverage information to each and every state of the lattice. Unlike the work presented in this thesis, they do not try to alleviate the cost of incorporating the language model, and only experiment with 3-gram LMs.[7]

[7]The use of Lagrangian multipliers, particularly in dealing with overlapping phrases, is not incompatible with our approach based on rejection sampling. However, that possibility is left for future work.

Chapter 4

Exact inference in SMT with OS*

In SMT, *optimisation* — the task of searching for an optimum translation — is performed over a high-complexity discrete distribution defined by the intersection between a translation hypergraph and a target language model. As discussed in Chapter 3, this distribution is too complex to be represented exactly and one typically resorts to approximation techniques such as beam-search (Koehn et al., 2003) and cube-pruning (Chiang, 2007), where maximisation is performed over a pruned representation of the full distribution.

Often, rather than finding a single optimum, one is really interested in obtaining a set of probabilistic *samples* from the distribution. This is the case in MBR decoding where one chooses from a sample of likely translations the one that minimises a loss function. Samples are also necessary in training where one needs to obtain unbiased estimates of expectations in order to fit the parameters of the model. Due to the additional computational challenges posed by sampling, n-best lists, a by-product of optimisation, are typically used as a biased approximation to true probabilistic samples. A known issue with n-best lists is that they tend to be clustered around the mode of the distribution (Blunsom and Osborne, 2008). A more effective procedure is to attempt to directly draw samples from the underlying distribution rather than relying on n-best list approximations (Arun et al., 2009, 2010).

MCMC methods, such as Gibbs sampling (Geman and Geman, 1984), offer a way to overcome the tractability issues in sampling, however their convergence properties are hard to assess (Gelman et al., 2013). That is, it is difficult to know when, if ever, an MCMC sampler produces samples that are compatible with the goal distribution. MCMC samples are dependent on each other in that they form a Markov chain. This means that it might take a long while before the samples can be assumed to provide a close approximation to the target distribution.

MC methods, such as rejection sampling (Robert and Casella, 2004), are more fundamental and natural, they can offer strong guarantees, but they are typically intractable. For instance, in rejection sampling, samples might be accepted very rarely, that is, the sampler has a very low acceptance rate, which ultimately implies prohibitively high running times.

A recent technique that stresses a unified view between the two types of inference tasks discussed here — optimisation and sampling — is the OS* approach (Dymetman et al., 2012a). OS* can be seen as a cross between adaptive rejection sampling (an MC method) and A* optimisation. In this view, rather than resorting to pruning in order to cope with the tractability issues, one upperbounds the complex goal distribution with a simpler "proposal" distribution for which dynamic programming is feasible. This proposal is incrementally refined to be closer to the goal until the maximum is found, or until the sampling performance exceeds a certain level.

In this thesis optimisation and sampling in SMT are done in an exact way by addressing search with the OS* algorithm. Exactness means that we are able to find the true optimum (in optimisation) and that we draw samples from the actual goal distribution (in sampling). In addition, at all times we have strong guarantees, such as, in optimisation it is always possible to know how far from the optimum we can be, and in sampling we have an unbiased estimate of the performance of the sampler (the acceptance rate).

The core of the technique involves lowering the complexity of the intersection between the translation hypergraph and the target language model. Rather than explicitly constructing the full intersection, which defines a distribution over derivations p, we incrementally produce a sequence of proposal hypergraphs that upperbound p, i.e., $q^{(0)} \geq q^{(t)} \geq \cdots \geq p$. This sequence is such that the first proposal $q^{(0)}$ is obtained by intersecting the translation hypergraph with a simple automaton that represents an optimistic low-order version of the full language model. Each proposal after that is obtained by intersecting the previous one with an automaton that incorporates some specific k-gram context not yet accounted for.

The idea of using OS* to lower the intersection between discrete weighted sets has been introduced before. Carter et al. (2012) apply OS* to sampling/optimisation with high-order HMMs in the context of **language modelling**. The high-order HMM corresponds to an intractable goal distribution (a wFSA) which is upperbounded by a sequence of tractable distributions for which optimisers and samplers can be obtained through standard dynamic programming techniques. Their work is related to the work presented in this thesis. However, note that in comparison to SMT, inference over language models is a simpler problem. For instance, in language modelling, words are produced from left to right with no reordering. Dymetman et al. (2012a) describe a **conceptual** approach to lower the complexity associated with the intersection between a probabilistic context-free grammar and a weighted finite-state automaton using OS*. Their work serves as inspiration to the work described in this thesis, particularly, in the context of hierarchical models. However, this thesis goes far beyond: we achieve a fully-working prototype of an OS* decoder/sampler coping with additional challenges not anticipated in that conceptual description. In addition, we extend the application of OS* to phrase-based SMT attempting to deal with NP-completeness.

The Contributions of this chapter are the following:

- a novel approach to exact inference in SMT based on lowering the complexity of the intersection between the translation hypergraph and the target language model (Section 4.2);

- an efficient computation of max-backoff weights on a sentence basis (Section 4.2.2.1);
- tight upperbounds on language model distributions (Section 4.2.2.2).

4.1 OS*

OS* (Dymetman et al., 2012b,a) is a framework for exact inference over high-dimensional spaces that proposes a unified view of optimisation and sampling. In the OS* approach these two inference tasks are seen as two extremes in a continuum of inference tasks in L^p spaces (Rudin, 1987), with sampling associated with the L^1 norm, and optimisation with the L^∞ norm.

The objective function p, over which inference needs to be performed, is a complex non-negative function over a discrete or continuous space X, which defines an unnormalised distribution over X. The goal is to optimise or sample relative to p, where sampling is interpreted in terms of the normalised distribution $\bar{p}(.) = p(.)/\int_X p(x)dx$.

Directly optimising or sampling from p is unfeasible; however, it is possible to define an unnormalised distribution q of lower complexity than p, which upperbounds p everywhere (i.e. $p(x) \leq q(x), \forall x \in X$), and from which it is feasible to optimise or sample directly.

4.1.1 Background

We revisit the formal definition of OS* introduced by Dymetman et al. (2012b,a) which is fundamental to the developments introduced in this thesis.

Let $p : X \to \mathbb{R}_+$ be a measurable function with respect to a base measure μ on a space X. The quantity $\|p\|_1 \equiv \int_X p(x)d\mu(x)$ is called the L^1 norm of p. We assume that $p \in L^1$, that is, $\|p\|_1$ is finite. We can think of p as an unnormalised density over X and $\bar{p}(x) \equiv \frac{p(x)}{\|p\|_1}$ its normalised version, which defines a probability distribution over X. The probability distribution $\bar{p}$ is the *goal distribution*, that is, the distribution over which one wishes to reason probabilistically.

Typically, for high-dimensional X, $\bar{p}$ is too complex to sample from. We assume, however, that $p(x)$ is easy to assess for any given $x \in X$. We proceed by *rejection sampling* (Robert and Casella, 2004) to obtain samples from p indirectly — by abuse of language when we refer to a sample from p we actually mean a sample from its normalised version $\bar{p}$. We define a lower-complexity unnormalised density q that *dominates* p, that is, $p(x) \leq q(x), \forall x \in X$. This upperbound density q is such that standard dynamic programming techniques are feasible and one can sample (or optimise) from it directly. In other words, q serves as a tractable proxy to p. The following procedure summarises rejection sampling, an MC technique that can be shown to produce exact samples from p (Robert and Casella, 2004):

1. draw a sample $x \in X$ from q;
2. compute the ratio $r(x) = \frac{p(x)}{q(x)}$, which lies in the interval $[0, 1]$ by construction;
3. with probability $r(x)$ accept x, otherwise reject it; and
4. repeat the process until a reasonable number of samples have been accepted.

This procedure produces samples from p at a rate given by $\frac{\|p\|_1}{\|q\|_1}$ known as the *acceptance rate*. However, due to p's complexity, $\|p\|_1$ is hard to assess. The average rate at which rejection sampling produces exact samples is an unbiased estimate of the true acceptance rate and it can be computed directly for a measurable subset A of X as $\frac{P(A)}{Q(A)}$, where $P(A) \equiv \int_A p(x) d\mu(x)$, and equivalently for Q. Figure 4.1 illustrates a goal density p and an upperbound proposal q. The sample x_1 is likely to be accepted because the ratio $r(x_1) = p(x_1)/q(x_1)$ is close to 1, while x_2 is likely to be rejected due to a low ratio (note how $q(x_2)$ is far above $p(x_2)$). The efficiency of the sampler illustrated

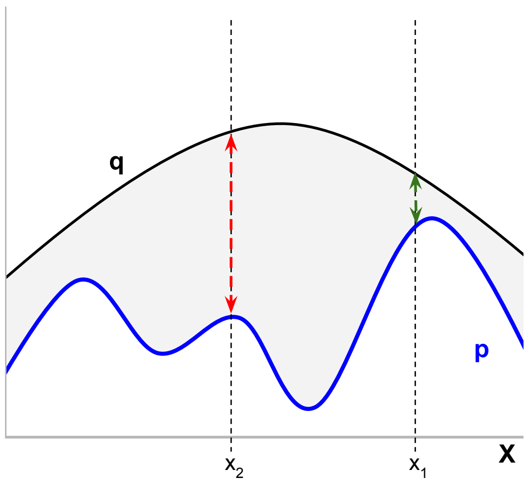


Figure 4.1: Rejection sampling

in Figure 4.1, otherwise known as its acceptance rate, is given by the ratio between the total areas below the curves p and q.

In the context of L^p spaces, sampling and optimisation can be viewed as two extremes of a continuous range of inference tasks. To elaborate on that we must first recollect a few definitions from measure theory. If (X, μ) is a measure space, and $f : X \to \mathbb{R}_+$ is a real-valued function on this space, the L^p norm of f, for $1 \leq p < \infty$, is defined as $\|f\|_p \equiv \left(\int_X |f|^p(x) d\mu(x)\right)^{1/p}$. The L^∞ norm of f is also defined and, for all practical purposes, it can be thought of as the maximum of f when it exists, i.e. $\|f\|_\infty = \max_{x \in X} |f|$.[1]

Dymetman et al. (2012b,a) introduced a generalisation of sampling based on L^p spaces (to avoid confusion with the goal distribution p, the notation L^p is changed to L^α). Under such generalisation one performs sampling of $p : X \to \mathbb{R}_+$ relative to $L^\alpha(X, \mu)$, for $1 \leq \alpha < \infty$, if one samples in the standard sense, that is, according to the normalised density $\bar{p}(x) \equiv \frac{p(x)^\alpha}{\|p\|_\alpha^\alpha}$. Additionally, one performs sampling of p relative to $L^\infty(X, \mu)$, for $\alpha = \infty$,

[1]A more formal definition of the L^∞ norm is left aside once the intuition that L^∞ corresponds to the maximum suffices the purposes of this thesis. The interested reader may consult (Dymetman et al., 2012b) for additional considerations or (Rudin, 1987) for a formal view.

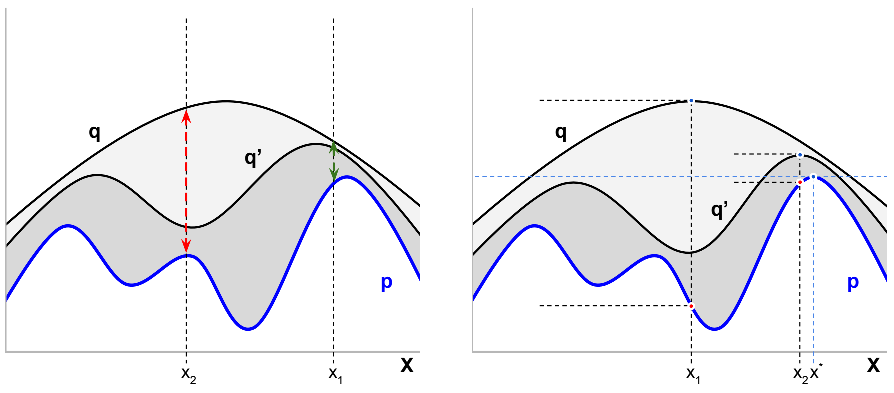
if one performs optimisation relative to p. The standard notion of sampling is relative to L^1 and optimisation is equivalent to sampling relative to L^∞. Sampling relative to L^α with large α "tends" to optimisation in that the values of p over samples relative to L^α become closer to the maximum of p as α increases. That is, as α increases, the samples relative to L^α become closer to the argmax of p.[2]

4.1.2 OS* sampling

In OS* sampling, first a sample x is drawn from q, and then x is accepted or rejected with probability given by the ratio $r = p(x)/q(x)$. Note that $0 \leq r \leq 1$ by construction. When a sample x from q is rejected, it is used as a basis for "refining" q into a slightly more complex q', where $p \leq q' \leq q$ is still an upperbound to p. This *adaptive rejection sampling* technique incrementally improves the rate of acceptance, and is pursued until some rate above a given threshold is obtained, at which point one stops refining and uses the current proposal to obtain further exact samples from p.

Figure 4.2a illustrates OS* sampling for a goal distribution over the one-dimensional real line. First the sample x_1 is accepted due to a high ratio $r(x_1)$. Then the sample x_2 is drawn and rejected due to a low ratio $r(x_2)$. At this point, information about x_2 is used to refine q into a q', such that $p(x) \leq q'(x) \leq q(x)$. Note that the proposal is brought closer to p leading to a better acceptance rate. The search may stop at this point if the acceptance rate has reached a certain threshold, or we continue sampling and refining on rejects.

[2]Such unified viewpoint is also implicitly shared by simulated annealing (Kirkpatrick et al., 1983) – an optimisation technique that capitalises on the idea of sampling with increasingly larger α.



(a) Sampling (b) Optimisation

Figure 4.2: OS* modes

4.1.3 OS* optimisation

Following the generalised idea of sampling, in optimisation one samples relative to L^∞. That is, one finds the maximum x relative to the proposal q, and again computes the ratio $r = p(x)/q(x)$. If this ratio equals 1, then it is easy to show that x is the actual maximum from p: if there existed x' such that $p(x') > p(x)$, then $q(x') \geq p(x') > p(x) = q(x)$, and hence x would not be a maximum for q, a contradiction. Otherwise the proposal is refined in a similar way to the sampling case, continuing until a ratio equal to 1 is found (or very close to 1, if one is willing to accept an approximation to the maximum). For finite spaces X, this optimisation technique is argued to be a generalisation of A* (Hart et al., 1968).[3]

Figure 4.2b illustrates OS* optimisation for a goal distribution over the one-dimensional real line. First x_1 is found as the maximum of q and rejected due to the low ratio $r(x_1)$, otherwise interpreted as a large gap between q and p at $x = x_1$. We refine the proposal obtaining q', whose maximum x_2 is lower than x_1, thus closer to the true maximum x^*. If we are willing to accept an approximate solution, and the gap between q' and p at x_2 is sufficiently

[3]Refer to (Dymetman et al., 2012b,a) for additional considerations.

small, i.e. $r(x_2)$ is sufficiently close to 1, the search may stop at this point, otherwise we proceed by refining the proposal.

4.1.4 Remarks

Refinement operations are similar in both modes of OS*, however they serve different purposes by aiming at different objectives.

In sampling the objective of the refinements is to minimise the L^1 norm of q', while making it only slightly more complex than q — always maintaining q' as an upperbound to p. In optimisation the objective is to minimise the L^∞ norm of q', while making it only slightly more complex than q — again always preserving q' as an upperbound to p. What these refinement operations look like and how they are chosen will be addressed in Section 4.2.3 when we introduce an OS* approach to SMT. Note that, intuitively, OS* sampling focuses the "search effort" on minimising the area between the proposal and the goal distribution, while OS* optimisation focuses on lowering the proposal's maximum so that it meets the goal.

4.2 An OS* approach to SMT

In this section we introduce a novel approach to exact inference in SMT using OS*.

Recapitulating from Section 2.4.1, $G(\mathbf{x})$ is the weighted set of all possible translations of the input $\mathbf{x}$.[4] At this point let us assume that $G(\mathbf{x})$ is tractable and acyclic.[5] Decoding requires the computation of $G(\mathbf{x}) \cap A$, where A is a

[4]For an input sentence $\mathbf{x}$, $G(\mathbf{x})$ is (ideally) obtained by the composition $(\mathcal{X} \circ \mathcal{G}) \downarrow$, where $\mathcal{X}$ is the identity-transducer of the set $\{\mathbf{x}\}$, $\mathcal{G}$ is a compact representation of the translation rules, and $\downarrow$ represents the output/target projection. In context-free models, $\mathcal{G}$ is a synchronous context-free grammar. In finite-state models, $\mathcal{G}$ is a cascade of finite-state transducers that encodes phrase segmentation, phrase translation and reordering. In both cases, $G(\mathbf{x})$ is equivalent to a hypegraph.

[5]Chapter 6 shows that for finite-state models, tractability is a too strong assumption in the general case due to arbitrary reordering.

finite-state automaton that represents an n-gram language model distribution over strings of the target language. It is also convenient to assume that for simple deterministic automata (with few states) an efficient intersection procedure is known — an argument that will be elaborated in Chapters 5 and 6.

The complexity of building the full intersection $G(\mathbf{x}) \cap A$ is related to the fact that the number of states of A grows exponentially with the order n of the language model. Chapter 5 shows that in the case that $G(\mathbf{x})$ is a weighted context-free grammar, $G_{\cap A} = G(\mathbf{x}) \cap A$ is also a context-free grammar, and that each nonterminal N in $G(\mathbf{x})$ tends to generate many indexed nonterminals of the form ${}_iN_j$ in the grammar $G_{\cap A}$, where i, j are states of A and the nonterminal ${}_iN_j$ can be interpreted as an N connecting an i state to a j state. A similar argument can be made for finite-state models.

In our approach, instead of explicitly constructing the full intersection $G(\mathbf{x}) \cap A$, which, using the notation of Section 4.1, is identified with the unnormalised goal distribution $p(\mathbf{d})$, we incrementally produce a sequence of "proposal" distributions $q^{(t)}$, which all upperbound p, where $q^{(0)} = G(\mathbf{x}) \cap A^{(0)}, \ldots, q^{(t+1)} = q^{(t)} \cap A^{(t)}$, etc. Here $A^{(0)}$ is an optimistic, low complexity, "unigram" version of the automaton A, and each increment $A^{(t)}$ is a small automaton that refines $q^{(t-1)}$ relative to some specific k-gram context (i.e. sequence of k words) not yet made explicit in the previous increments, where k takes some value between 1 and n. This process produces a sequence of proposals such that $q^{(0)}(\cdot) \geq q^{(1)}(\cdot) \geq q^{(2)}(\cdot) \geq \cdots \geq p(\cdot)$.

In the limit $\bigcap_{t=0}^{M} A^{(t)} = A$ for some large M, so that we are in principle able to reconstruct the full intersection $p(\cdot) = q^{(M)} = G(\mathbf{x}) \cap A^{(0)} \cap \cdots \cap A^{(M)}$ in finite time. In practice our actual process stops much earlier: in optimisation, when the value of the maximum derivation $\mathbf{d}^*$ relative to $q^{(t)}$ becomes equal to its value according to the full language model; in sampling when the

Algorithm 6 OS* for SMT

```
1:  $t \leftarrow 0$, converged $\leftarrow$ false          ▷ in sampling also does $AR \leftarrow 0$
2:  $q^{(0)} \leftarrow G(\mathbf{x}) \cap A^{(0)}$
3:  while not converged do
4:      $\mathbf{d} \leftarrow$ search$(q^{(t)})$          ▷ argmax or sample
5:      $r \leftarrow p(\mathbf{d})/q^{(t)}(\mathbf{d})$
6:      accept $\leftarrow$ assess$(r)$   ▷ deterministic in optimisation, random in sampling
7:      if not accept then          ▷ if $\mathbf{d}$ was rejected
8:          define $A^{(t+1)}$ based on $\mathbf{d}$ and $q^{(t)}$
9:          $q^{(t+1)} \leftarrow q^{(t)} \cap A^{(t+1)}$          ▷ update proposal
10:         $t \leftarrow t + 1$
11:     end if
12:     converged $\leftarrow$ update$(r, \text{accept})$    ▷ in sampling this also updates $AR$
13: end while
14: return accepted samples along with $q^{(t)}$
```

acceptance rate of samples from $q^{(t)}$ exceeds a certain threshold.

Algorithm 6 illustrates the application of OS* to SMT. In line 1, the time step is initialised to 0. In sampling the initial acceptance rate (AR) is also set to 0. In line 2, the initial proposal $q^{(0)}$ is the result of intersecting $G(\mathbf{x})$ with $A^{(0)}$.

In line 3 we start a loop: in optimisation we stop when we have found an x that is accepted, meaning that the maximum has been found; in sampling we stop when the estimated acceptance rate (AR) of the current proposal $q^{(t)}$ exceeds a certain threshold (e.g. 20%). This AR can be roughly estimated by observing how many of the last (say) one hundred samples have been accepted, and tends to reflect the actual acceptance rate obtained by using $q^{(t)}$ without further refinements. These monitors are updated at the end of each iteration, in line 12.

In line 4, in optimisation, we find the derivation $\mathbf{d}$ that maximises the proposal, and in sampling we draw a sample $\mathbf{d}$ from the proposal.[6] In line 5,

[6]See in Section 4.1.1 that, following the OS* approach, taking an *argmax* is actually associated to an extreme form of sampling, with an L^∞ space taking the place of an L^1 space. We may also sample with $\alpha > 1$, exploring other L^α spaces — for the reader familiar with simulated annealing, α is the inverse of the temperature and controls how

we compute the ratio $r = \frac{p(\mathbf{d})}{q^{(t)}(\mathbf{d})}$. Remember that by construction $q^{(t)}$ is an optimistic version of p, thus $0 \leq r \leq 1$.

In line 6, in optimisation we accept $\mathbf{d}$ if the ratio r is equal to 1, in which case we have found the maximum, and in sampling we accept $\mathbf{d}$ with probability r.[7] This form of adaptive rejection sampling guarantees that accepted samples form exact samples from p (Robert and Casella, 2004).

If $\mathbf{d}$ is rejected (line 7), we (lines 8 and 9) refine $q^{(t)}$ into a $q^{(t+1)}$ such that $p(\cdot) \leq q^{(t+1)}(\cdot) \leq q^{(t)}(\cdot)$ everywhere. This is done by defining the incremental automaton $A^{(t+1)}$ on the basis of x and $q^{(t)}$ (see Section 4.2.3), and by intersecting this automaton with $q^{(t)}$.

Finally, in line 14, in optimisation we return the $\mathbf{d}$ which has been accepted, namely the maximum of p, and in sampling we return the list of already accepted $\mathbf{d}$'s, which form an exact sample from p. We also return the current $q^{(t)}$, which can be used directly in sampling as a rejection sampler to produce further exact samples with an acceptance rate performance above the predefined threshold. In optimisation the current $q^{(t)}$ can be used to produce an n-best list. This n-best list can be approximate, in case one searches for the n-best solutions from $q^{(t)}$ without further refinements. Or it can be exact: (i) first remove the current best solution from $q^{(t)}$ (via intersection); then (ii) restart the loop in line 3 of Algorithm 6, iterating until the the next best solution is proven; and (iii) repeat steps (i) and (ii) until the exact n-best solutions have been found.

4.2.1 Upperbound on LM distribution

Let p_{lm} be an n-gram language model distribution and w an upperbound to p_{lm}, that is, $w(\cdot) \geq p_{lm}(\cdot)$ everywhere. The distribution w can be defined in

flat/peaked the distribution is.

[7]In optimisation, we control the search with a parameter $0 < \epsilon \leq 1$, accepting $\mathbf{d}$ if $r \geq \epsilon$. Note that, if $\epsilon = 1$, we iterate until the true maximum is found. Otherwise, if we are willing to accept an approximate solution, we can choose $0 < \epsilon < 1$.

terms of p_{lm} as shown in Equation 4.1, where (P, z), also denoted by Pz, is a string of n words in the vocabulary Σ of the language model, z is the string's last word and P is z's prefix of length $(n-1)$ words.

$$w(z|P) = \max_{h \in \Sigma^*} p_{lm}(z|hP) \tag{4.1}$$

The term $p_{lm}(z|hP)$ is the conditional probability of the n-gram (hP, z). Maximisation happens over all possible histories, that is, all possible sequences of words that may precede Pz in a sentence. Intuitively, $w(z|P)$ assigns to the string Pz the LM probability which is that of Pz in its most promising context.

The computation of p_{lm} for a string of arbitrary length is a recursive procedure that factorises in terms of the entries in a standard ARPA table for n-gram language models (Jurafsky and Martin, 2000). In an ARPA table, n-grams are stored in ascending order of length (lower-order n-grams first). Each entry in the table is a tuple consisting of (1) an n-gram string Pz, (2) the n-gram's conditional probability denoted by $Pz.p$, and (3) the n-gram's backoff weight denoted by $Pz.b$. Given an ARPA table T, the conditional probability $p_{lm}(z|P)$ for an arbitrary sequence Pz is computed as shown in Equation 4.2, where **tail**(P) is the string resulting of the deletion of P's first word. Note that P's backoff weight $P.b$ is used when the entry Pz is not explicitly listed in the table, but the prefix P is.

$$p_{lm}(z|P) = \begin{cases} p_{lm}(z|\text{tail}(P)) & \text{if } Pz \notin T \text{ and } P \notin T \\ p_{lm}(z|\text{tail}(P)) \times P.b & \text{if } Pz \notin T \text{ and } P \in T \\ Pz.p & \text{if } Pz \in T \end{cases} \tag{4.2}$$

In the case of a language model of maximum order n, the probability $w(z|P)$ is sensitive to all sequences of up to $n-1$ words. In the worst case, the maximisation in Equation 4.1 would require applying the recursive Equation 4.2 a number of times proportional to $O(|\Sigma|^{n-1})$. Carter et al.

Algorithm 7 Max-ARPA: first pass

```
1: for $P$ in $T$ do
2:     $P.m \leftarrow 1$
3:     for $x$ in $\Sigma$ s.t $xP$ in $T$ do
4:         $P.m \leftarrow \max(P.m, xP.b \times xP.m)$
5:     end for
6: end for
```

(2012) noticed that w factorises in terms of something they called *max-backoff weights* allowing a fast computation of w for a string of arbitrary length, shown in Equation 4.3. They designed an efficient offline computation of these weights from a standard ARPA table and stored the result in what they called a "Max-ARPA" table, an extension of the original format that accommodates two additional factors: an optimistic view on the n-gram conditional probability denoted by $Pz.w$, and an optimistic view on the backoff weight denoted by $Pz.m$. These two factors can be computed in two passes over the standard ARPA table in descending order of n-gram length (higher-order n-grams first), as shown in Algorithms 7 and 8.[8]

$$w(z|P) = \begin{cases} p_{lm}(z|P) & \text{if } Pz \notin M \text{ and } P \notin M \\ p_{lm}(z|P) \times P.m & \text{if } Pz \notin M \text{ and } P \in M \\ Pz.w & \text{if } Pz \in M \end{cases} \tag{4.3}$$

In the first pass the optimistic backoff weight is computed and stored for each entry P. In the second pass the optimistic conditional probability is computed for each entry Pz in T. Note that maximisation happens over the 1-word context $x \in \Sigma$, i.e., all the words in the vocabulary covered by the language model. Also, in the second pass, if a sequence xPz is not listed in the table, the algorithm backs off to xP.

Equation 4.3 shows the efficient computation of w factorised in terms of the max-backoff weights and p_{lm}. This computation requires at most one

[8]Algorithm due to Marc Dymetman, personal communication.

Algorithm 8 Max-ARPA: second pass

```
1:  for $Pz$ in $T$ do
2:      $Pz.w \leftarrow Pz.p$
3:      for $x$ in $\Sigma$ s.t $xP$ in $T$ do
4:          if $xPz$ in $T$ then
5:              $Pz.w \leftarrow \max(Pz.w, xPz.w)$
6:          else
7:              $Pz.w \leftarrow max(Pz.w, Pz.p \times xP.b \times xP.m)$
8:          end if
9:      end for
10: end for
```

call to Equation 4.2, which is resolved in at most $|P|$ recursions. This is a significant improvement compared to the worst case of Equation 4.1, which would require $O(|\Sigma|^{n-1})$ calls to Equation 4.2 (which is itself recursive). Note that if the sequence Pz is listed in the Max-ARPA table M, the value $Pz.w$ is directly returned, since it represents the probability of Pz in its most promising context. Otherwise, if Pz is not in M, but P is, the algorithm resorts to $p_{lm}(z|P)$ and scales it by the optimistic view of P's backoff weight. Finally, if neither Pz nor its prefix P are listed in M, there is no reason to look for an optimistic view on the string and $p_{lm}(z|P)$ is returned.[9] A formal proof that this procedure complies with the specification that w dominates p_{lm} is presented in (Dymetman, 2013).

4.2.2 Initial proposal

The initial proposal $q^{(0)}$ is the result of intersecting the translation hypergraph $G(\mathbf{x})$ with an automaton that represents an optimistic view of the full language model A. This initial automaton $A^{(0)}$ is a deterministic automaton, and it is simpler than A in that it only records information about unigrams. $A^{(0)}$ has only one state q_0, which is both initial and final. For each word y

[9]This is only true when the original ARPA table respects the property that if an n-gram is in the table, all of its substrings are also in the table. For certain LM pruning strategies that do not guarantee this property, the table must be extended to account for the missing substrings explicitly.

of the target language it has a transition $\langle q_0, y, q_0 \rangle$ whose weight is denoted by $w(y)$.[10]

An important observation that leads to tighter max-backoff weights is that we can restrict histories — in Algorithms 7 and 8 — to words that appear in $G(\mathbf{x})$, that is, rather than assuming that every word in the target vocabulary might precede z we consider only a subset of those made of unigrams that are compatible with the rules in $G(\mathbf{x})$.

A tighter upperbound means that the initial proposal $q^{(0)}$ is closer to p, which in turn means that the area between the two surfaces is smaller. In sampling this means that we start at a better acceptance rate. In optimisation, $q^{(0)}$'s maximum is closer to p's maximum and we are left with a smaller gap to be incrementally reduced.

On the one hand we start from a better proposal, on the other hand the Max-ARPA computation depends on $G(\mathbf{x})$ and can no longer be precomputed once from the full ARPA table. We propose an efficient algorithm to compute the Max-ARPA table on a sentence basis (Section 4.2.2.1) tightening the initial proposal based on the active vocabulary of unigrams in $G(\mathbf{x})$. We also propose an efficient algorithm to tighten the initial proposal based on the active vocabulary of bigrams in $G(\mathbf{x})$ (Section 4.2.2.2).

4.2.2.1 Sentence basis Max-ARPA

An ARPA table can be compactly and conveniently represented in memory or in disk as a *trie* (Germann et al., 2009). A trie (Fredkin, 1960), or prefix tree, is a tree-structured transition graph that can be seen as a deterministic finite-state automaton. Each node of a trie is associated with a unique string given by the concatenation of the labels in the transitions along the unique path from the root to the node in consideration. The string along the path can be seen as a key and the node may store a value. Thus, a trie is a well-

[10]We use $w(P, y)$ to denote the conditional probability $w(y|P)$. In cases where P is the empty string ϵ, we write $w(y)$.

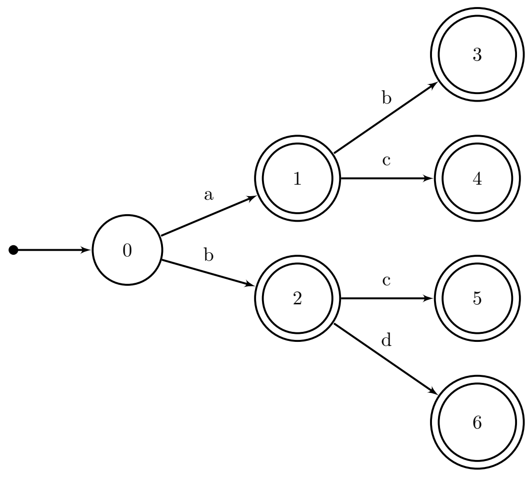


Figure 4.3: Example of a trie

established data structure to compactly store key-value pairs where the keys often share prefixes. Insertion and lookup are proportional to $O(m)$ where m is the length of the key. Figure 4.3 illustrates a small trie, the initial state is indicated with an arrow and final states are represented by double circles. For example, state 5 represents the string *bc* and state 6 represents the string *bd*.

A trie-encoded ARPA table can be thought of as a tree in which nodes (or states) represent n-grams. The root, the initial state of the trie, represents the empty string ϵ. Transitions are labelled with words in the vocabulary Σ of the language model, therefore, states have a one-to-one correspondence with n-grams, i.e. entries in the table.[11] Final states contain information about the n-grams they represent, such as the n-gram probability and its backoff weight. Moreover, in language modelling, it is convenient to perform lookup in suffix order, thus strings are inserted in reversed order (Stolcke, 2002; Federico et al., 2008; Heafield, 2011). Tries used like that are often called "reverse tries".

In order to efficiently compute a Max-ARPA table whose weights are

[11]For unpruned language models, every state in a trie — not only the leaves — is final, except maybe for the initial one.

Algorithm 9 Filtering an ARPA table to a set of active unigrams.

```
function QUERY(prefix, U, T, LM)
    E ← T[prefix]            ▷ entry in the trie that corresponds to the prefix
    if E not None then
        if E.final then       ▷ final states contain information about $n$-grams
            LM[prefix] = (E.p, E.b)                                ▷ ARPA weights
        end if
        C ← E.outgoing                     ▷ all 1-word continuations of the prefix
        for c in U ∩ C do                           ▷ retain only the active words
            QUERY(prefix + [c], U, T, LM)               ▷ query the extended prefix
        end for
    end if
end function
function FILTER(U,T)
    LM ← MemoryTrie()
    unk ← Set()
    for x in V do
        if x in T.root.outgoing then
            QUERY([x], U, T, LM)
        else
            unk.add(x)
        end if
    end for
    return LM, unk
end function
```

tighter in that they are computed on the basis of the active vocabulary of unigrams U in $G(\mathbf{x})$, we can collect from the ARPA table only the n-grams that are compatible with U and then rely on Algorithms 7 and 8 in Section 4.2.1. Algorithm 9 shows an efficient procedure to collect from a large ARPA table T, encoded as a trie in disk (with efficient lookup algorithms), a smaller ARPA table LM, encoded as a trie in memory, that contains all n-grams compatible with U, that is, n-grams whose words are all in U. The algorithm visits the states of the trie that are compatible with U copying them to the memory trie. The number of states visited is bounded by the maximum number of n-grams compatible with strings in $G(\mathbf{x})$, that is, $|U|^n$.

Algorithm 10 Filtering an ARPA table to a set of active bigrams.

```
1:  function QUERY(prefix, U, B, T, LM)
2:      E ← T[prefix]                        ▷ entry in the trie that corresponds to the prefix
3:      if E not None then
4:          if E.final then                  ▷ final states contain information about $n$-grams
5:              LM[P] = (E.p, E.b)                                           ▷ ARPA weights
6:          end if
7:          C ← E.outgoing                         ▷ all 1-word continuations of the prefix
8:          for c in U ∩ C do                                ▷ retain only the active words
9:              last ← prefix[-1]
10:             if (last,c) in B then                       ▷ retain only the active bigrams
11:                 QUERY(prefix + [c], U, B, T, LM)          ▷ query extended prefix
12:             end if
13:         end for
14:     end if
15: end function
```

4.2.2.2 Tighter proposals

We can further improve the tightness of our proposal by relying not only on the set of active unigrams U in $G(\mathbf{x})$, but also on the set of active bigrams. The strings in $G(\mathbf{x})$ are compatible with a **subset** of the bigrams in $U \times U$. While in principle this subset could actually coincide with the whole set $U \times U$, it is typically much smaller, meaning that for any given unigram $z \in U$, the set of possible 1-word contexts such that hz is compatible with a string in $G(\mathbf{x})$ is much smaller than U. Suppose that B is the set of all bigrams that can participate in strings of $G(\mathbf{x})$. Then the recursive call to QUERY in line 9 of Algorithm 9 could be conditioned on (h, c) belonging to B, as shown in Algorithm 10 (line 10).

Sections 5.4 and 6.5 discuss how we compute B for context-free and finite-state models, respectively.

4.2.3 Incremental refinements

The weight assigned to any target sentence by $A^{(0)}$ is larger or equal to its weight according to A. Recall that A is the automaton associated with

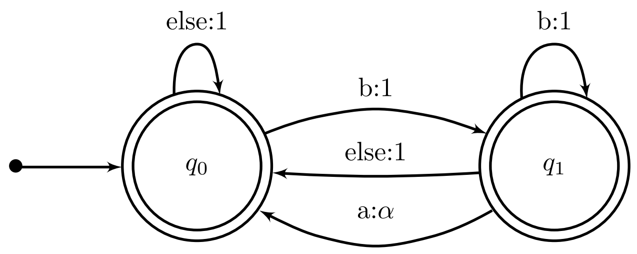


Figure 4.4: Down-weighting a in the context of b

the full language model distribution. Therefore, the initial proposal $q^{(0)} = G(\mathbf{x}) \cap A^{(0)}$ is *optimistic* relative to the goal $p = G(\mathbf{x}) \cap A$. That is, for any solution $\mathbf{d}$ in p, we have $p(\mathbf{d}) \leq q^{(0)}(\mathbf{d})$. We can then apply the OS* technique with $q^{(0)}$. In the case of **optimisation**, this means that we find the solution $\mathbf{d}$ for which $q^{(0)}$ is maximum. By construction, with $\mathbf{y} = \mathrm{y}[\mathbf{d}]$ (the string that $\mathbf{d}$ projects onto), we have $A^{(0)}(\mathbf{y}) \geq A(\mathbf{y})$. If the two values are equal, we have found the maximum,[12] otherwise there must be a word y_i in the sequence $\mathbf{y} = y_1^m$ for which $p_{lm}(y_i|y_1^{i-1})$ is strictly smaller than $w_1(y_i)$.[13] Let us take among such words the one for which the ratio $\alpha = w_2(y_i|y_{i-1})/w_1(y_i) \leq 1$ is the smallest, and for convenience let us rename $b = y_{i-1}, a = y_i$. We then define the (deterministic) automaton $A^{(1)}$ as illustrated in Figure 4.4.

In $A^{(1)}$, the state q_0 is both initial and final, and the state q_1 is final. All edges carry a (multiplicative) weight equal to $\bar{1}$, except edge $\langle q_1, a, q_0 \rangle$, which carries the weight α. The abbreviation "else" refers to any label other than b when starting from q_0, and other than b or a when starting from q_1.

It is easy to check that this automaton assigns to any word sequence $\mathbf{y}$ a weight equal to α^k, where k is the number of occurrences of $b\,a$ in $\mathbf{y}$. In particular, if $\mathbf{y}$ is such that $y_{i-1} = b, y_i = a$, then the transition in (the deterministic automaton) $A^{(0)} \cap A^{(1)}$ that consumes y_i carries the weight $\alpha\, w_1(a)$, in other words, the weight $w_2(a|b)$. Thus the new proposal $q^{(1)} =$

[12]This case is very unlikely with $A^{(0)}$, but helps introduce the general case.
[13]Here the index n in $w_n(\sigma)$ highlights the order of the n-gram σ.

$q^{(0)} \cap A^{(1)}$ has now "incorporated" knowledge of the bigram a-in-the-context-b, at the cost of some increase in its complexity.[14]

The general procedure for choosing $A^{(t+1)}$ follows the same pattern. We find the maximum $\mathbf{d}$ in $q^{(t)}$ along with its yield $\mathbf{y}$. If $p(\mathbf{d}) = q^{(t)}(\mathbf{d})$, we stop and output $\mathbf{d}$. Otherwise, we find some subsequence $y_{i-m-1}, y_{i-m}, ..., y_i$ such that the knowledge of the n-gram $y_{i-m}, ..., y_i$ has already been registered in $q^{(t)}$, but not that of the n-gram $y_{i-m-1}, y_{i-m}, ..., y_i$ (this bookkeeping is explained in Section 4.2.4), and we define an automaton $A^{(t+1)}$ which assigns to a sequence a weight α^k, where

$$\alpha = \frac{w_{m+1}(y_i|y_{i-m-1}, y_{i-m}, ..., y_{i-1})}{w_m(y_i|y_{i-m}, ..., y_{i-1})} \tag{4.4}$$

and where k is the number of occurrences of $y_{i-m-1}, y_{i-m}, ..., y_i$ in the sequence.[15]

We note that we have $p \leq q^{(t+1)} \leq q^{(t)}$ everywhere, and also that the number of possible refinement operations is bounded. This is so because at some point we would have expanded all contexts to their maximum order, at which point we would have reproduced $p(\cdot)$ on the whole space $\mathcal{D}$ of possible solutions exactly. However, we typically stop much earlier than that, without expanding contexts in the regions of $\mathcal{D}$ which are not promising even on optimistic assessments based on limited contexts.

Following the OS* methodology, the situation with **sampling** is completely analogous to that of optimisation. However, instead of finding the solution $\mathbf{d}$ for which $q^{(t)}$ is maximum, we draw a sample $\mathbf{d}$ from the dis-

[14]Note that without further increasing $q^{(1)}$'s complexity one can incorporate knowledge about all bigrams sharing the prefix b. This is because $A^{(1)}$ does not need additional states to account for different continuations of the context b. All one needs is to update the weights of the transitions leaving state 1 appropriately. More generally, it is not more costly to account for all 1-word continuations of a k-gram prefix than it is to account for only one of them.

[15]Building $A^{(t+1)}$ is a variant of the standard construction for a "substring-searching" automaton (Cormen et al., 2001) and produces an automaton with n states (the order of the n-gram).

tribution associated with $q^{(t)}$, then accept it with probability given by the ratio $r = p(\mathbf{d})/q^{(t)}(\mathbf{d}) \leq 1$. In the case of a reject, we identify a subsequence $y_{i-m-1}, y_{i-m}, ..., y_i$ in $\mathrm{y}[\mathbf{d}]$ as in the optimisation case, and similarly, refine $q^{(t)}$ into $q^{(t+1)} = q^{(t)} \cap A^{(t+1)}$. The acceptance rate gradually increases because $q^{(t)}$ comes closer and closer to p. We stop the process at a point where the current acceptance rate, estimated on the basis of, say, the last one hundred trials, exceeds a predefined threshold, such as 20%.[16]

4.2.4 Bookkeeping

A key operation in OS* is the update, or refinement, of the current proposal $q^{(t)}$. As discussed in earlier sections, we upperbound the language model distribution A starting from an optimistic unigram LM distribution $A^{(0)}$. That is, $A^{(0)}$ is a proposal LM distribution. To avoid confusion, when we refer to "the proposal" we mean the hypergraph proposal $q^{(t)}$ used as proxy to the goal distribution p. On the other hand, when we refer to "the proposal LM" we mean an implicit automaton that represents the current state of the upperbound language model distribution, and we denote it as A_0^t. Note that at some iteration t, $q^{(t)} = G(\mathbf{x}) \cap A^{(0)} \cap A^{(1)} \cap \cdots \cap A^{(t)}$, and $A_0^t = \bigcap_{i=0}^t A^{(t)}$ is the implicit intersection of all refinements since the beginning of the search. We stress the word "implicit" because in practice the upperbound weights are incorporated directly in the hypergraph via incremental intersections.

In Section 4.2.3, it was explained that a refinement $A^{(t)}$ incorporates more realistic n-gram weights. It lowers the upperbound by expanding some k-gram context not yet incorporated in the proposal LM A_0^{t-1}. In order to identify which contexts have not yet been accounted for, one needs to keep track of the contexts that have been incorporated in past iterations since

[16]Instead of designing $A^{(t+1)}$ on the basis of a single sample derivation, we may sample a batch of (say) one hundred samples. This way we gather better statistics about overoptimistic n-grams that are likely to participate in derivations sampled from the current grammar.

the beginning of the search. This *bookkeeping* is performed in a very simple manner utilising a trie to compactly store information about the n-grams incorporated in the search. In a way, this trie gives a view of the proposal LM. Formally, the trie is not equivalent to the automaton that represents $A^{(0)} \cap A^{(1)} \cap \cdots \cap A^{(t)}$. However, it does contain the information about the n-grams that have been explicitly represented in the proposal LM and their (upperbound) weights. One can think of this trie as an ARPA table, however, this ARPA table is optimistic with respect to the true language model and it initially factorises in terms of unigrams. As the algorithm proceeds incorporating some higher-order n-grams, this trie evolves to represent a variable-order LM proposal distribution.

The trie used to bookkeep n-gram information over a vocabulary Σ is equivalent to a tree-structured deterministic automaton $T = \langle \Sigma, Q, I, F, E\rangle$. It has a single initial state, $I = \{q_0\}$, and every state in the trie is final, $F = Q$. E is an unweighted set of transitions such that the set of paths from the trie's initial state q_0 to any given state $q \in F$, $P(q_0, q) \subseteq E^*$, is a singleton. Thus, there is a one-to-one correspondence between paths and strings, meaning that the set of all valid paths $P(I, x, F)$ that recognise a string $x \in \Sigma^*$ is also a singleton. Therefore, each state $q \in Q$ is associated to a unique string in Σ^* recognised by the path from q_0 to q. To bookkeep the information we need, each and every state $q \in Q$ is associated with a k-gram conditioning context $\alpha \in \Sigma^k$ given by the *reverse* of the string recognised by the unique path from q_0 to q. Moreover, each state stores a function that maps a word $z \in \Sigma$ to the upperbound weight $w(z|\alpha)$, otherwise denoted by $w_{k+1}(\alpha z)$. Formally, if $\pi = \langle e_1, e_2, \ldots, e_k\rangle$ is a path in the trie, where e_i is a transition, the path's origin $p[\pi] = q_0$ is the trie's initial state, and $\alpha = \langle i[e_k], i[e_{k-1}], \ldots, i[e_1]\rangle$ is the reverse of the string recognised by π, then the path's destination state $n[\pi]$ holds the function $w_{k+1}(\alpha z)$ that weights all 1-word continuations $z \in \Sigma$ of the k-gram context α.

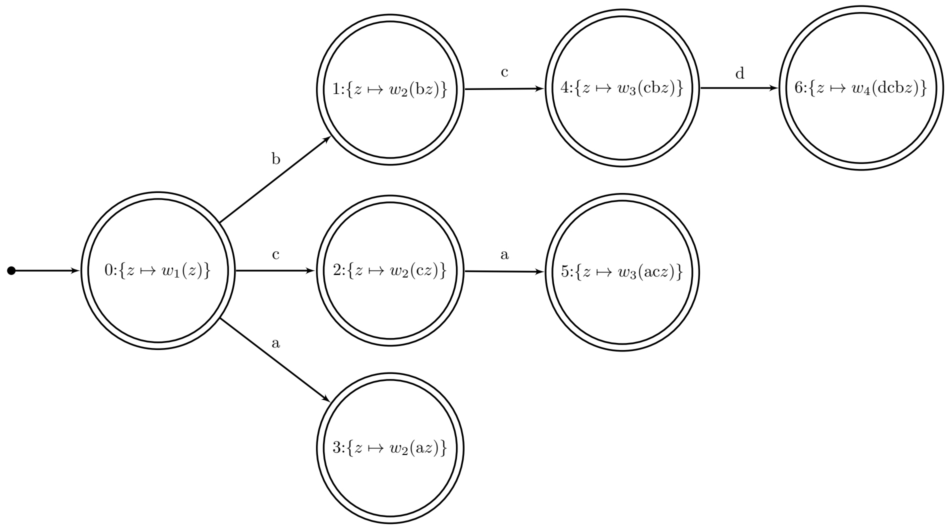

0:{$z \mapsto w_1(z)$}
1:{$z \mapsto w_2$(bz)}
2:{$z \mapsto w_2$(cz)}
3:{$z \mapsto w_2$(az)}
4:{$z \mapsto w_3$(cb$z$)}
5:{$z \mapsto w_3$(ac$z$)}
6:{$z \mapsto w_4$(dcb$z$)}
b
c
a
c
a
d

Figure 4.5: Trie used to bookkeep the n-grams in the proposal LM

The initial state stores upperbound weights of words conditioned on an empty context, that is, the unigrams. Thus, the trie associated with A_0^0 contains a single state, which is initial and final, and stores the unigram upperbound weights $w_1(z)$. Figure 4.5 illustrates a trie used to bookkeep n-gram information over the vocabulary $\Sigma = \{a, b, c, d, e\}$. It stores all the 1-word continuations of the k-gram contexts ϵ, a, b, c, ac, cb and dcb, as illustrated by the mappings inside of the nodes. In OS*, one starts from a unigram proposal LM and incorporate higher-order n-grams as necessary leading to a variable order representation where n-grams are always extended to the left. Therefore, it is convenient to represent *reversed contexts* in the trie. At any moment, given a derivation $\mathbf{d}$ whose yield $\mathbf{y}$ is the string $y_1y_2 \dots y_m$, one can quickly retrieve the longest suffix of $y_1y_2 \dots y_{m-1}$ represented in the trie by first reversing it $y_{m-1}y_{m-2} \dots y_1$ and then recognising the longest prefix (of this reversed suffix) that is compatible with the trie.

4.3 Experiments

For the experiments reported in this section we used the German-English portion of the 6th version of the Europarl collection (Koehn, 2005). In all experiments, we used the Moses toolkit (Koehn et al., 2007) to extract a wSCFG following Chiang (2005).[17] Language models were trained by *lmplz* (Heafield et al., 2013) using the English monolingual data made available by the WMT (Callison-Burch et al., 2012). That is, *Europarl*, *newscommentaries*, *news-2012* and *commoncrawl*. The models extracted from different monolingual corpora were combined via interpolation using *newstest2010* as the development set. In all experiments, parameter estimation was performed via MERT on the *newstest2010* development set using single reference BLEU as the oracle. Finally, from the *newstest2011* test set, we randomly selected sentences according to their length. We sampled 20 examples for each class of length from 1 to 30 words.

In this section we compare three strategies to obtain an upperbound on a bigram LM distribution. One strategy is to produce a Max-ARPA table directly from a complete ARPA table, which represents a form of **corpus** level computation. Recall that a Max-ARPA table factorises the computation shown in Equation 4.1 in terms of max-backoff weights as shown in Equation 4.3. This table is obtained once and used *as is* for every sentence. Another strategy is to produce a Max-ARPA table for each sentence on the basis of the active target vocabulary in the grammar $G(\mathbf{x})$. The intuition is that by considering only a subset of the target vocabulary in the full LM, the upperbound obtained should be tighter. A tighter upperbound on the LM distribution leads to q being a closer approximation to q and ultimately reduces the number of iterations for convergence to an optimum or to a minimum acceptance rate level. In the following, we tighten the upperbound

[17]We extracted grammars containing at most two nonterminals on the right-hand side and at most 10 target productions for a given source production.

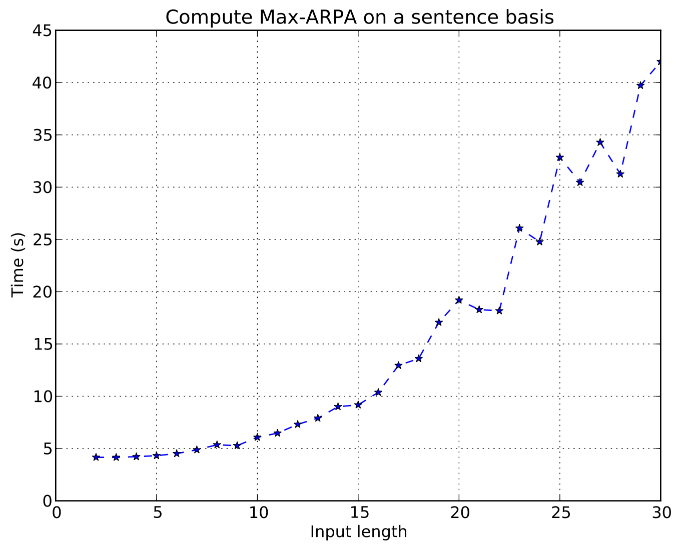


Figure 4.6: Sentence basis Max-ARPA

on the basis of the vocabulary of active **unigrams** in $G(\mathbf{x})$ which we denote by U. Because we have efficient algorithms to enumerate the bigrams in $G(\mathbf{x})$ (see Sections 5.4 and 6.5), we can also go one step further and limit the Max-ARPA computation to n-grams compatible with the vocabulary B of **bigrams** in $G(\mathbf{x})$.

Computing a Max-ARPA table for each sentence requires filtering an ARPA table to the active target vocabulary associated with each sentence's translation hypergraph. A naïve strategy is to visit each and every entry of the ARPA table retaining only the n-grams compatible with U. Such algorithm runs in time proportional to the size of the ARPA table, which is inefficient for any reasonably sized language model. The complexity of our solution is bounded by the maximum number of n-grams in the filtered table, that is, $|U|^n$ (see Section 4.2.2.1). Figure 4.6 shows the average time necessary to build a Max-ARPA table on a sentence basis as a function of the input length. Observe how the time appears to grow quadratically with the length of the sentence. This is the case because the size of the

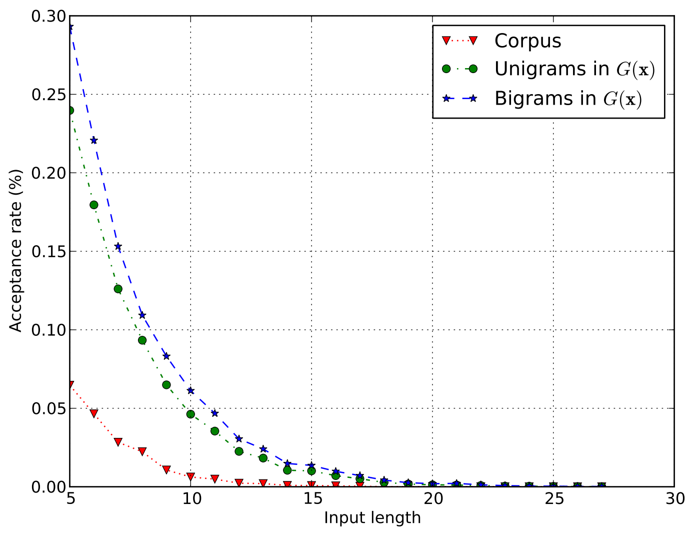


Figure 4.7: Initial acceptance rate with different upperbounds

vocabulary is typically linear with the input length and filtering the ARPA table of a bigram language model (see Algorithm 9) runs in time $O(|U|^2)$. The construction in Figure 4.6 involves: (1) running Algorithm 9 to obtain a small "active ARPA" table whose n-grams contain only words in U; and (2) computing max-backoff weights as shown in Algorithms 7 and 8.

Figure 4.7 shows how the different upperbounds affect the acceptance rate of the initial proposal. We can clearly see that restricting the Max-ARPA computation to the active vocabulary in $G(\mathbf{x})$ improves the tightness of the upperbound. There is a large gain moving from no filtering at all (**corpus**) to considering the set of **unigrams**. There is a modest gain moving from **unigrams** to **bigrams**. Beyond 20 words the initial acceptance rate is negligible in all cases. However, tighter upperbounds do not only increase the initial acceptance rate, they also lead to fewer iterations before convergence to a goal because the proposal q starts closer to p. For example, Figure 4.8 shows how much each upperbound overestimates the score of the optimum derivation $\mathbf{d}^*$. The y-axis represents the gap (in the log domain) between

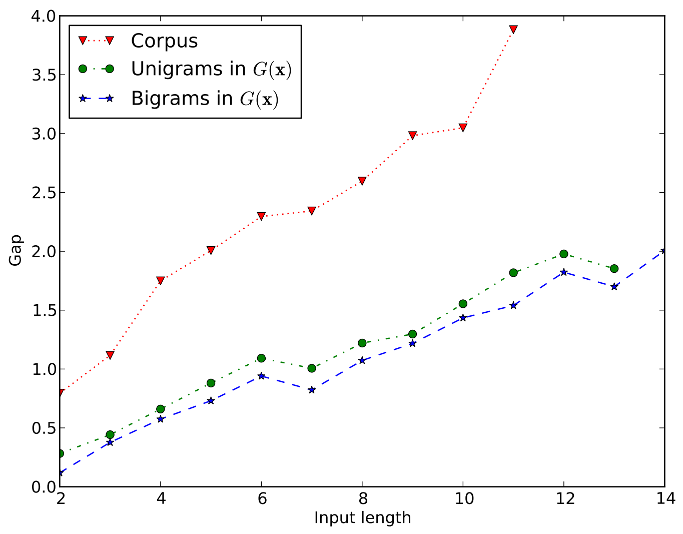


Figure 4.8: Gap (in the log domain) between $p(\mathbf{d}^*)$ and $q^{(0)}(\mathbf{d}^*)$ with different upperbounds

$p(\mathbf{d}^*)$, the true score of the true optimum, and $q^{(0)}(\mathbf{d}^*)$, the true optimum assessed by the initial proposal distribution. We can clearly see that tightening the upperbound to the vocabulary of the grammar helps lowering the gap between p and q.

Chapter 5

Hierarchical phrase-based translation with OS*

In this chapter we introduce an OS* approach to exact inference for hierarchical phrase-based translation models such that of Chiang (2005). For convenience, we first introduce the hierarchical case to avoid having to deal at first with NP-completeness. The phrase-based case is introduced in Chapter 6.

Based on OS* (Dymetman et al., 2012b,a) and our application of OS* to SMT presented in Section 4.2, the contributions we present in this chapter are the following:[1]

- a novel approach to **exact inference** (optimisation and sampling) for hierarchical phrase-based translation models (Section 5.2);
- an algorithm for **incremental intersection** between wCFGs and wFSAs (Section 4.2.3). This algorithm capitalises on the incremental aspect of the refinements that incorporate additional context to the proposals;
- an algorithm to **selectively intersect** a wCFG with a wFSA expanding the grammar only in high-probability regions by taking into account the marginal probability of the productions in the original grammar (Section 5.3.2).

[1]An earlier and more concise version of this chapter was published at the 8th Workshop on Statistical Machine Translation (Aziz et al., 2013).

5.1 Hierarchical phrase-based SMT

In hierarchical phrase-based translation, the underlying model is represented by a wSCFG (Chiang, 2005, 2007). That is, $\mathcal{G}$ in $G(\mathbf{x}) = (\mathcal{X} \circ \mathcal{G}) \downarrow$ (see Section 2.5) is a wSCFG. The translation hypergraph $G(\mathbf{x})$ (also known as a translation forest) is obtained by applying $\mathcal{G}$ to the input sentence, in other words, by the composition (a generalisation of intersection for transducers, see Section 2.1.3.3) between the input $\mathbf{x}$, represented as the identity-transducer $\mathcal{X}$, and the synchronous grammar $\mathcal{G}$. Because $\mathcal{X}$ is acyclic and our hierarchical model does not allow unbounded insertion, the resulting grammar $G(\mathbf{x})$ is also acyclic (Chiang, 2005). Moreover, limiting productions in $\mathcal{G}$ to having up to two nonterminals on their right-hand side, the number of rules in $G(\mathbf{x})$ is proportional to $O(I^3)$, where I is the number of words in the input sentence (Wu, 1995).

Decoding requires re-weighting the translations in $G(\mathbf{x})$ by the language model A. The space of solutions is given by the intersection $G_{\cap A} = G(\mathbf{x}) \cap A$, which is itself a wCFG and represents the goal distribution p over which we want to perform inference. The intersection between a wCFG and a wFSA has polynomial theoretical bounds, but is nevertheless prohibitive to compute exactly due to mostly two reasons. First, the number of states of A grows exponentially with the order n of the language model. Second, each nonterminal N in $G(\mathbf{x})$ tends to generate many indexed nonterminals of the form ${}_iN_j$ in the grammar $G_{\cap A}$, where i and j are states of A and the nonterminal ${}_iN_j$ can be interpreted as an N connecting state i to state j.

The goal distribution $p(\mathbf{d}) = G(\mathbf{x}) \cap A$ is typically intractable even for language models of order 2 and one commonly resorts to approximations such as cube-pruning (Chiang, 2007). In Section 5.2, we introduce a novel approach to exact inference over p using OS*. This approach enables exact optimisation and exact sampling by capitalising on a sequence of tractable proxies to p.

5.2 An OS* approach to HPB-SMT

We introduce an approach where instead of explicitly constructing the full intersection $G(\mathbf{x}) \cap A$, which is identified with the unnormalised goal distribution $p(\mathbf{d})$, we incrementally produce a sequence of "proposal" grammars $q^{(t)}$. These proposals are such that $q^{(0)} = G(\mathbf{x}) \cap A^{(0)}$, $q^{(1)} = q^{(0)} \cap A^{(1)}$, …, $q^{(t+1)} = q^{(t)} \cap A^{(t+1)}$, etc. $A^{(0)}$ is an optimistic, low complexity, "unigram" version of the automaton A. Each increment $A^{(t)}$ is a small automaton that incorporates to $q^{(t-1)}$ some specific k-gram context (i.e. sequence of k words) not yet made explicit in the previous increments, where k takes some value between 1 and n. This process produces a sequence of proposals such that $q^{(0)}(\cdot) \geq q^{(1)}(\cdot) \geq q^{(2)}(\cdot) \geq \cdots \geq p(\cdot)$.

For this to work it is essential that an efficient intersection procedure is known. We introduce a novel incremental intersection algorithm based on "Earley intersection" (Dyer, 2010). Our procedure capitalises on the incremental aspect of the refinements.

To achieve faster intersections it is necessary to control the size of the grammar. This can be done by carefully intersecting the grammar with the automaton, not everywhere, but rather in regions that are likely to participate in high-scoring derivations. We introduce a new algorithm that explores information from the current grammar, such as the rule marginal probabilities, to perform a selective intersection. In this selective intersection, additional context is incorporated only in high-probability regions of the grammar. That is, regions that are low-probability even on optimistic assessments are left at a lower-order representation with respect to the n-gram language model. Note that this does not characterise as pruning, since no edges are removed from the proposal hypergraphs. Instead, they are simply left at a lower-order representation, and they are only expanded when proven necessary.

5.3 Intersecting grammars and automata

In their classical paper, Bar-Hillel et al. (1961) showed that the intersection of a CFG with an FSA is a CFG. Their construct is an exhaustive procedure in which rules from the input CFG are annotated with pairs of states that can be connected through a path in the input FSA. That is, context-free rules of the form $X \rightarrow \alpha_1 \dots \alpha_k$, where $X \in V$ and $\alpha_i \in \Sigma \cup V$ are annotated with states $q_i \in Q$ producing ${}_{q_0}X_{q_k} \rightarrow {}_{q_0}{\alpha_1}_{q_1}\, {}_{q_1}{\alpha_2}_{q_2} \cdots {}_{q_{k-1}}{\alpha_k}_{q_k}$ for all possible sequences of states $q_0 \dots q_k$. A symbol of the kind ${}_qX_r$, with $X \in V$ and $q, r \in Q$, represents a nonterminal X spanning a path in the FSA from q to r. In addition, for each symbol ${}_qx_r$ in a rule, where x is a terminal, if $\langle q, x, r\rangle$ is a transition in the input FSA, the annotated symbol ${}_qx_r$ is replaced by the terminal x. Otherwise, the rule can be deleted since it is not compatible with the FSA.

Observe that there are $|Q|^r$ sequences of states of length r, which makes this a very inefficient algorithm. Although inefficient, this construct makes evident that the intersected grammar has rules that are very similar to the original ones. They are identical in length and structure differing only in that the nonterminals are annotated so as to memorise paths in the FSA.

Billot and Lang (1989) were possibly the first to notice the connection of this construct with chart-parsing. In general, parsing with a CFG can be seen as a special case of intersection, with the input sequence represented as a "flat" (linear-chain) automaton. This insight allows to generalise various parsing algorithms to corresponding intersection algorithms. One such algorithm for wCFG and wFSA inspired by the CKY parsing algorithm was introduced by Nederhof and Satta (2008b). In their procedure, nonterminals are annotated in a bottom-up phase that starts from valid spans in the wFSA, followed by a top-down pruning that discards rules that can never participate in a valid derivation. Additionally, when a terminal symbol x is incorporated by a rule, by substitution of its annotated version ${}_q x_r$, the weight of the associated rule is $\otimes$-multiplied by the weight of the corresponding wFSA transition $\langle q, x, r\rangle$.

Dyer and Resnik (2010) introduced an efficient top-down algorithm for intersecting a wCFG with a wFSA. The algorithm is a generalisation of the Earley algorithm for parsing (Earley, 1970). The advantage of Dyer's "Earley intersection" algorithm is that it combines top-down predictions with bottom-up completions. The algorithm thus avoids constructing many nonterminals that may be justified from the bottom-up perspective, but can never be "requested" by a top-down derivation, and would need to be pruned in a second pass.[2] Figure 5.1 is a compact representation of Dyer's Earley intersection using a weighted deductive proof system (Pereira and Warren, 1983; Shieber et al., 1995; Goodman, 1998, 1999). This program accepts as input an epsilon-free wCFG $G = \langle \Sigma, V, \langle S, \sigma\rangle, \langle R, \nu\rangle\rangle$ and an epsilon-free wFSA $A = \langle \Sigma, Q, \langle I, \lambda\rangle, \langle F, \rho\rangle, \langle E, w\rangle\rangle$.[3]

A deductive program progresses by combining axioms and premises under certain conditions to prove a goal. The bracketed expressions are called items. When they appear above the horizontal line they are called antecedents. When they appear below the horizontal line they are called consequent. The side expression is a condition that enables the deduction. In Earley intersection, items are dotted rules, where the dot represents the progress of the intersection and is associated to a state in the automaton. For example, the interpretation of an item such as $[X \to \alpha \bullet Y\beta, q, s] : u$, with q and s states in Q, X and Y nonterminals in V, and $X \to_u \alpha Y \beta$ a rule in R, where $\alpha, \beta \in (\Sigma \cup V)^*$, is that we are trying to match a nonterminal X spanning

[2]Our early experiments showed an important gain in intermediary storage and in overall time by using this Earley-based technique as opposed to a CKY-based technique.

[3]Dyer (2010) introduces a slightly more general procedure that handles ϵ-rules and ϵ-transitions.

$$\frac{}{[S' \to \bullet X, q, q] : \lambda(q) \otimes \sigma(X)} \quad (q \in I) \wedge (X \in S) \qquad \textsc{Axioms}$$

$$[S' \to X \bullet, q, r] \quad (q \in I) \wedge (r \in F) \wedge (X \in S) \qquad \textsc{Goals}$$

$$\frac{[X \to \alpha \bullet Y\beta, q, r] : u}{[Y \to \bullet \gamma, r, r] : v} \quad Y \to_v \gamma \in R \qquad \textsc{Predict}$$

$$\frac{[X \to \alpha \bullet z\beta, q, s] : u}{[X \to \alpha z \bullet \beta, q, r] : u \otimes w(s, z, r)} \quad \langle s, z, r\rangle \in E \qquad \textsc{Scan}$$

$$\frac{[X \to \alpha \bullet Y\beta, q, s] : u \; [Y \to \gamma \bullet, s, r] : v}{[X \to \alpha \, {}_sY_r \bullet \beta, q, r] : u} \quad X \neq S' \qquad \textsc{Complete}$$

$$\frac{[S' \to \bullet X, q, q] : u \; [X \to \gamma \bullet, q, r] : v}{[S' \to {}_qX_r \bullet, q, r] : u \otimes \rho(r)} \quad (r \in F) \wedge (X \in S) \qquad \textsc{Accept}$$

Figure 5.1: Logic program to perform Earley intersection

from state q and that we have succeeded in matching the part α, spanning from state q to state s, and we are now trying to match β spanning from s. An item whose dot happens before the end of the rule is said to be incomplete. A complete item is such as $[Y \to \gamma \bullet, s, r] : v$, which represents a Y spanning from state s to state r with weight v, and ${}_rY_s$ (sometimes also written $Y_{r,s}$) is a shorthand for the resulting left-hand side nonterminal. The number of items in the program is bounded by $|R||Q|^r$, where r is the length of the longest right-hand side in R plus one, that is, the length of the sequence of states that indexes the longest rule in the grammar.

The **axioms** of the program shown in Figure 5.1 consist of all items expecting to prove that a start symbol of the grammar can recognise paths starting from an initial state of the automaton. To make the procedure general enough to handle multiple start symbols, multiple initial states and multiple final states, the axiom introduces a new start symbol S', which will become the root of the intersected grammar. Hence, the **goal** is to prove that start symbols span valid paths in the automaton, that is, paths from an initial to a final state. **Prediction** is an operation that creates new items from a dotted rule whose dot precedes a nonterminal, under the condition that this nonterminal is associated to productions in the grammar. **Scan** is an operation that moves the dot forward over a terminal symbol, provided that this symbol can be recognised in the automaton from the state represented by the dot. Note that for deterministic automata, scanning produces a single item per deduction. Scanning causes the incorporation, by $\otimes$-multiplication, of the weights in wFSA transitions. **Completion** is used to *merge* items, (a) one that expects a nonterminal X spanning paths from a certain state s, and (b) another proving that X spans paths from s to some other state r; in the resulting item the dot moves forward over the nonterminal proving paths up to r. Finally, complete items are **accepted**. A complete item such as $[Y \to \gamma \bullet, s, r] : v$ produces a rule ${}_sY_r \to_v \gamma$ in the intersected grammar $G' = \langle \Sigma, V', \langle \{S'\}, \{S' \mapsto \bar{1}\}\rangle, \langle R', \nu' \rangle\rangle$, where V' is the specialised set of nonterminals, S' is the new start symbol, and $\langle R', \nu'\rangle$ is the specialised set of rules.

To illustrate how the deductive program works, consider the wCFG

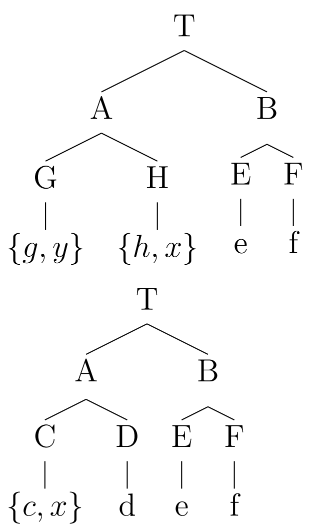
$$\begin{aligned} q^{(0)} = \langle \quad & \Sigma = \{c, d, e, f, g, h, x, y\}, \\ & V = \{A, B, C, D, E, F, G, H, T\}, \\ & \langle \{T\}, \{T \mapsto \bar{1}\}\rangle, \\ & \langle R_0, u \rangle \qquad\qquad\qquad\qquad \rangle \end{aligned}$$

shown in Figure 5.2, where Figure 5.2a shows the weighted set of rules $\langle R_0, u\rangle$ and Figure 5.2b compactly illustrates the forest that R_0 encodes ($q^{(0)}$ contains 6 derivations).[4] Figure 5.3 shows an example refinement $A^{(1)}$ that lowers the upperbound score of derivations containing bigrams starting in x. Figure 5.3a shows the wFSA used to locate the bigrams and update their scores,

[4]We use a compact notation which we call *packed forest* to group derivations identical in structure, however yielding different strings. Alternative terminals are grouped at the leaves using a set notation.

1	$\mathrm{T} \xrightarrow[u_1]{} \mathrm{A\ B}$
2	$\mathrm{B} \xrightarrow[u_2]{} \mathrm{E\ F}$
3	$\mathrm{A} \xrightarrow[u_3]{} \mathrm{G\ H}$
4	$\mathrm{A} \xrightarrow[u_4]{} \mathrm{C\ D}$
5	$\mathrm{H} \xrightarrow[u_6]{} \mathrm{h}$
6	$\mathrm{H} \xrightarrow[u_5]{} \mathrm{x}$
7	$\mathrm{C} \xrightarrow[u_7]{} \mathrm{x}$
8	$\mathrm{C} \xrightarrow[u_8]{} \mathrm{c}$
9	$\mathrm{E} \xrightarrow[u_9]{} \mathrm{e}$
10	$\mathrm{F} \xrightarrow[u_{10}]{} \mathrm{f}$
11	$\mathrm{G} \xrightarrow[u_{11}]{} \mathrm{g}$
12	$\mathrm{G} \xrightarrow[u_{12}]{} \mathrm{y}$
13	$\mathrm{D} \xrightarrow[u_{13}]{} \mathrm{d}$

(a) Rules (R_0)

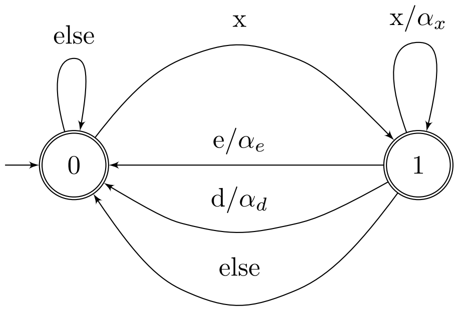


(b) Forest

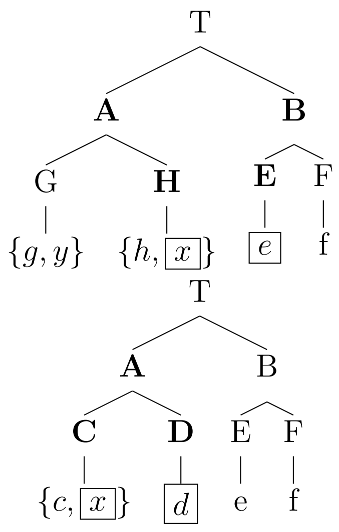
Figure 5.2: Example $q^{(0)} = G(\mathbf{x})$

there, $\alpha_z = {}^{w_2(xz)}/_{w_1(z)}$ refers to the scaling factor that updates the score of the unigram z to the bigram xz (see Section 4.2.3). Figure 5.3b highlights the bigrams $\{xe, xd\}$ we would like to account for.

Figure 5.4 illustrates the intersection between $q^{(0)}$ (Figure 5.2) and the automaton $A^{(1)}$ (Figure 5.3a) following the deductive program in Figure 5.1. The explanation relies on a graphical illustration of the items of the deductive program. For instance, Figure 5.4a is associated with the item $[S' \rightarrow \bullet T, 0, 0] : \lambda(0) \otimes \sigma(T)$, where the new start symbol S' is omitted for the sake of space and the weights are omitted for the sake of clarity. Similarly, Figure 5.4b is associated with the item $[T \rightarrow \bullet A\, B, 0, 0]$ and so on. The illustration shows how the intersected grammar is built by making the indexed nonterminals explicit within the items. Incomplete nonterminals are denoted by ${}_{q_i}X_{\bullet}$, where $q_i \in \{0, 1\}$ is the origin state and $\bullet$ refers to the fact that we do not yet know the destination state of the paths spanned by X. Such symbols fire *predictions* in case they have never been predicted before.



(a) Refinement



(b) Interactions

Figure 5.3: Example refinement $A^{(1)}$

They can also be *completed*, in case some ${}_{q_i}X_{q_f}$ has been proven, causing the dot to move forward in the item. A terminal to be *scanned* is denoted by ${}_{q_i}z_\bullet$. Similarly, ${}_{q_i}\{z_1, \ldots, z_n\}_\bullet$ is used to compactly denote n items about to scan their respective terminals departing from state q_i.

Table 5.1 explains Figure 5.4 panel by panel. The packed forests in Figures 5.4h and 5.4j represent points were the intersection procedure "branched" the grammar. That is, it moved from recognising terminals at the intial state to recognising terminals at some other state which implicitly guarantees that a certain prefix has been recognised. In this running example, at state 1 it is certain that a prefix ending in x has been recognised. In Table 5.1, the items involved in the intersection are explained as well as how they are combined creating new items. A concrete implementation of the intersection procedure does not typically run in the order shown in Table 5.1, that order was chosen to comply with Figure 5.4.

Figure 5.5 shows the resulting grammar $q^{(1)} = q^{(0)} \cap A^{(1)}$, its modified set of rules R_1 (5.5a) and its derivations (5.5b) where the additional top rule

$S' \underset{\bar{1}}{\rightarrow} T$ is omitted. The intersection procedure creates 6 additional rules ($^+$) and it also reweights existing ones (*). Note that the wFSA weights are $\bar{1}$ except for $w(1, d, 0) = \alpha_d$ and $w(1, e, 0) = \alpha_e$ (Figure 5.3a).

$$
\begin{aligned}
q^{(1)} = \langle \quad & \Sigma = \{c, d, e, f, g, h, x, y\}, \\
& V = \{A, A', B, B', C, C', D, D', E, E', F, G, H, H', T\}, \\
& \langle \{T\}, \{T \mapsto \bar{1}\} \rangle, \\
& \langle R_1, v \rangle \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \rangle
\end{aligned}
$$

The intersected grammar contains the same six derivations of the input, however, the intersection requires an extended set of nonterminals, and additional rules to allow for derivations compatible with bigrams xz to be weighted more realistically. One can think of the refinement operation as a mechanism to make explicit, through the structure of the grammar, the derivations whose yield are compatible with the strings reweighted by the automaton, separating them from others that are not. That is basically what the boxed forests are highlighting in Figure 5.5b.

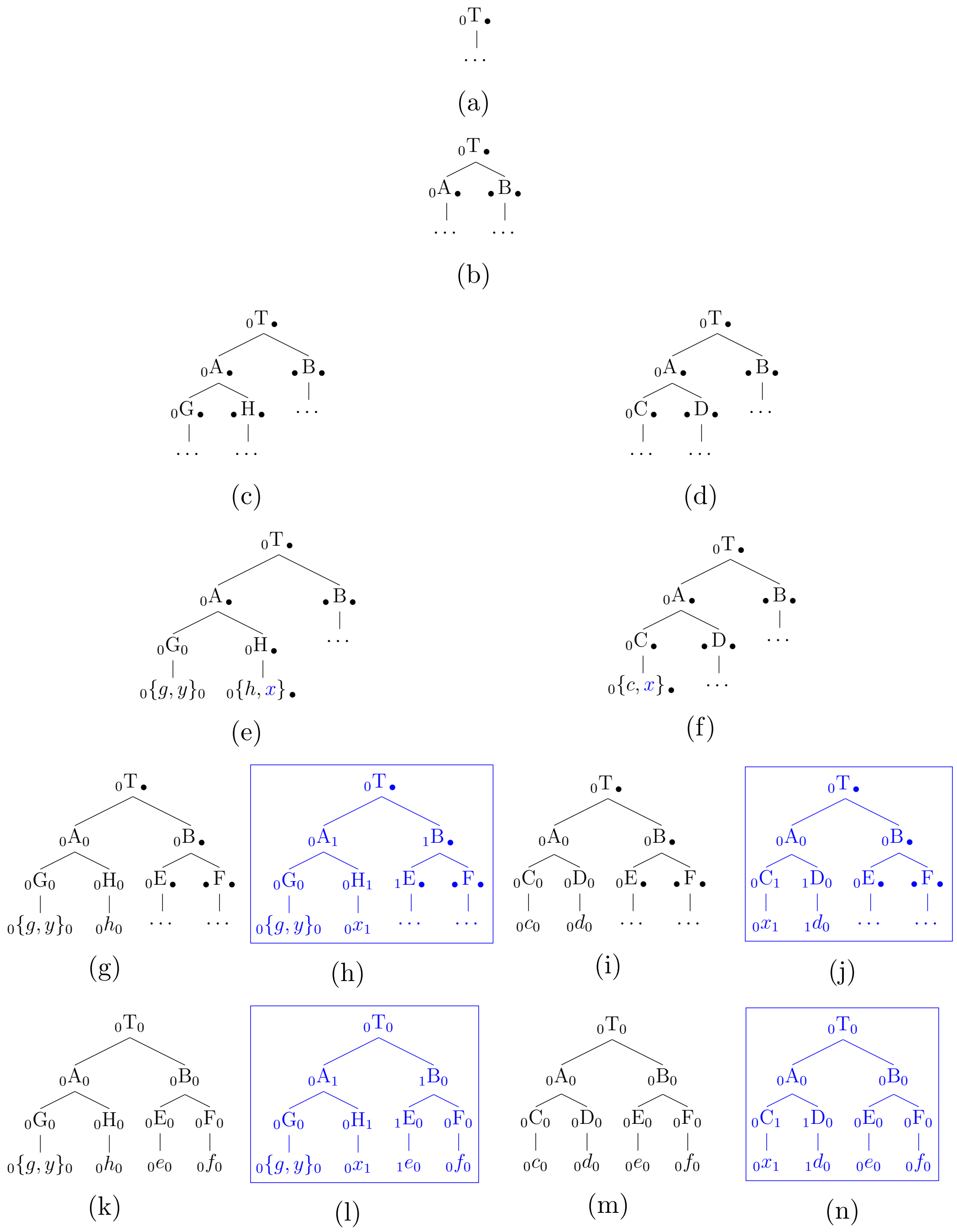


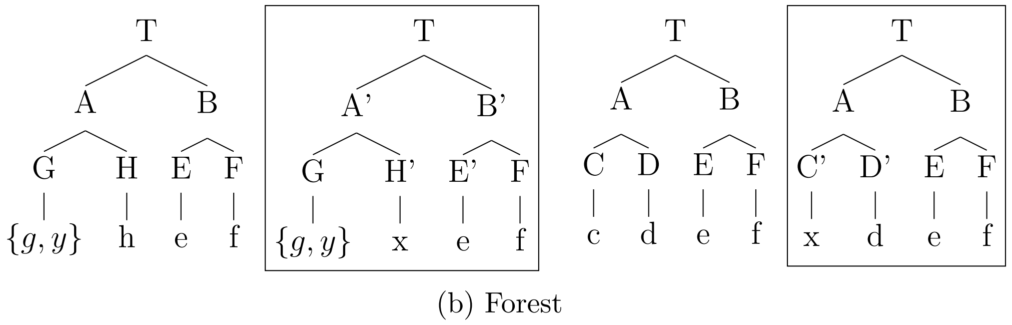
Figure 5.4: Intersecting the bigram $\langle x \bullet \rangle$ everywhere in $G(\mathbf{x})$

Figure	**Item**		**Explanation**
5.4a	1	$[S' \to \bullet T, 0, 0]$	this item is an axiom
5.4b	2	$[T \to \bullet A\, B, 0, 0]$	predicted from 1
5.4c	3	$[A \to \bullet G\, H, 0, 0]$	predicted from 2
5.4d	4	$[A \to \bullet C\, D, 0, 0]$	predicted from 2
5.4e	5	$[G \to \bullet g, 0, 0]$	predicted from 3
	6	$[G \to \bullet y, 0, 0]$	predicted from 3
	7	$[G \to g \bullet, 0, 0]$	scanned from 5 (proves $G_{0,0}$)
	8	$[G \to y \bullet, 0, 0]$	scanned from 6
	9	$[A \to G_{0,0} \bullet H, 0, 0]$	completed 3 with 7
	10	$[H \to \bullet h, 0, 0]$	predicted from 9
	11	$[H \to \bullet x, 0, 0]$	predicted from 9
5.4f	12	$[C \to \bullet c, 0, 0]$	predicted from 4
	13	$[C \to \bullet x, 0, 0]$	predicted from 4
5.4g	14	$[H \to h \bullet, 0, 0]$	scanned from 10 (proves $H_{0,0}$)
	15	$[A \to G_{0,0}\, H_{0,0} \bullet, 0, 0]$	completed 9 with 14 (proves $A_{0,0}$)
	16	$[T \to A_{0,0} \bullet B, 0, 0]$	completed 2 with 15
	17	$[B \to \bullet E\, F, 0, 0]$	predicted from 16
5.4h	18	$[H \to x \bullet, 0, 1]$	scanned from 11 (proves $H_{0,1}$)
	19	$[A \to G_{0,0}\, H_{0,1} \bullet, 0, 1]$	completed 9 with 18 (proves $A_{0,1}$)
	20	$[T \to A_{0,1} \bullet B, 0, 1]$	completed 2 with 19
	21	$[B \to \bullet E\, F, 1, 1]$	predicted from 20
5.4i	22	$[C \to c \bullet, 0, 0]$	scanned from 12 (proves $C_{0,0}$)
	23	$[A \to C_{0,0} \bullet D, 0, 0]$	completed 4 with 22
	24	$[D \to \bullet d, 0, 0]$	predicted from 23
	25	$[D \to d \bullet, 0, 0]$	scanned from 24 (proves $D_{0,0}$)
5.4j	26	$[C \to x \bullet, 0, 1]$	scanned from 13 (proves $C_{0,1}$)
	27	$[A \to C_{0,1} \bullet D, 0, 1]$	completed 4 with 26
	28	$[D \to \bullet d, 1, 1]$	predicted from 27
	29	$[D \to d \bullet, 1, 0]$	scanned from 28 (proves $D_{1,0}$)
5.4k	30	$[E \to \bullet e, 0, 0]$	predicted from 17
	31	$[E \to e \bullet, 0, 0]$	scanned from 30 (proves $E_{0,0}$)
	32	$[B \to E_{0,0} \bullet F, 0, 0]$	completed 17 with 31
	33	$[F \to \bullet f, 0, 0]$	predicted from 32
	34	$[F \to f \bullet, 0, 0]$	scanned from 33 (proves $F_{0,0}$)
	35	$[B \to E_{0,0}\, F_{0,0} \bullet, 0, 0]$	completed 32 with 34 (proves $B_{0,0}$)
	36	$[T \to A_{0,0}\, B_{0,0} \bullet, 0, 0]$	completed 16 with 35 (goal)
5.4l	37	$[E \to \bullet e, 1, 1]$	predicted from 21
	38	$[E \to e \bullet, 1, 0]$	scanned from 37 (proves $E_{1,0}$)
	39	$[B \to E_{1,0} \bullet F, 1, 0]$	completed 21 with 38
	40	$[B \to E_{1,0}\, F_{0,0} \bullet, 1, 0]$	complete 39 with 34 (proves $B_{1,0}$)
	41	$[T \to A_{0,1}\, B_{1,0} \bullet, 0, 0]$	completed 20 with 40 (goal)

Table 5.1: Earley intersection explained

i	**Item**	**Rule**	**i**	**Item**	**Rule**
1	${}_0T_0 \xrightarrow[u_1]{} {}_0A_0\ {}_0B_0$	T $\xrightarrow[v_1]{}$ A B	7*	${}_0C_1 \xrightarrow[u_7 \otimes w(0,x,1)]{}$ x	C' $\xrightarrow[v_7]{}$ x
+14	${}_0T_0 \xrightarrow[u_1]{} {}_0A_1\ {}_1B_0$	T $\xrightarrow[v_{14}]{}$ A' B'	8*	${}_0C_0 \xrightarrow[u_8 \otimes w(0,c,0)]{}$ c	C $\xrightarrow[v_8]{}$ c
2	${}_0B_0 \xrightarrow[u_2]{} {}_0E_0\ {}_0F_0$	B $\xrightarrow[v_2]{}$ E F	9*	${}_0E_0 \xrightarrow[u_9 \otimes w(0,e,0)]{}$ e	E $\xrightarrow[v_9]{}$ e
+15	${}_1B_0 \xrightarrow[u_2]{} {}_1E_0\ {}_0F_0$	B' $\xrightarrow[v_{15}]{}$ E' F	+18	${}_1E_0 \xrightarrow[u_9 \otimes w(1,e,0)]{}$ e	E' $\xrightarrow[v_{18}]{}$ e
3	${}_0A_0 \xrightarrow[u_3]{} {}_0G_0\ {}_0H_0$	A $\xrightarrow[v_3]{}$ G H	10*	${}_0F_0 \xrightarrow[u_{10} \otimes w(0,f,0)]{}$ f	F $\xrightarrow[v_{10}]{}$ f
+16	${}_0A_1 \xrightarrow[u_3]{} {}_0G_0\ {}_0H_1$	A' $\xrightarrow[v_{16}]{}$ G H'	11*	${}_0G_0 \xrightarrow[u_{11} \otimes w(0,g,0)]{}$ g	G $\xrightarrow[v_{11}]{}$ g
4	${}_0A_0 \xrightarrow[u_4]{} {}_0C_0\ {}_0D_0$	A $\xrightarrow[v_4]{}$ C D	12*	${}_0G_0 \xrightarrow[u_{12} \otimes w(0,y,0)]{}$ y	G $\xrightarrow[v_{12}]{}$ y
+17	${}_0A_0 \xrightarrow[u_4]{} {}_0C_1\ {}_1D_0$	A $\xrightarrow[v_{17}]{}$ C' D'	13*	${}_0D_0 \xrightarrow[u_{13} \otimes w(0,d,0)]{}$ d	D $\xrightarrow[v_{13}]{}$ d
5*	${}_0H_0 \xrightarrow[u_6 \otimes w(0,h,0)]{}$ h	H $\xrightarrow[v_6]{}$ h	+19	${}_1D_0 \xrightarrow[u_{13} \otimes w(1,d,0)]{}$ d	D' $\xrightarrow[v_{19}]{}$ d
6*	${}_0H_1 \xrightarrow[u_5 \otimes w(0,x,1)]{}$ x	H' $\xrightarrow[v_5]{}$ x			

(a) Rules (R_1)



(b) Forest

Figure 5.5: Intersected grammar $q^{(1)}$

5.3.1 Incremental intersection

The deductive program is a specification, its implementation may vary, but the result is the same. It requires that all possible deductions are performed, and that no deduction is performed twice with the same premises, which can be done by employing some standard form of agenda. Typically in a deductive program, items are managed in an agenda without duplicates. Items are said *active* if they are still to be operated upon, and *passive* otherwise. Active items are generated from other active items or by combination of active and passive items by exhaustive application of the operators of the program (e.g. scan, prediction, completion) until no active item is left.

We introduce here a variant of Dyer's Earley intersection that is tailored for simple automata of the kind discussed in Section 4.2.3 (recursive automata with few states). This variant is motivated by the fact that our refining automata consume most strings at the initial state with no change in weight. This means that for strings incompatible with the refining context (typically, the majority) the intersection with an $A^{(t)}$ will copy the structure of the original grammar. When computing $q^{(t)} = q^{(t-1)} \cap A^{(t)}$, rather than letting the standard algorithm reconstruct the part of the agenda that is directly associated with the input grammar $q^{(t-1)}$, our variant anticipates those items prior to the standard Earley intersection.

Observe that in Figure 5.5 the grammar $q^{(1)}$ is the result of intersecting the grammar $q^{(0)}$ in Figure 5.2a with the automaton $A^{(1)}$ (Figure 5.3a), that is, the automaton that reweights bigrams starting in x (denoted by xz). The derivations compatible with xz are the boxed ones. Note that for the other derivations the grammar looks just like the original.

The intuition is to perform a single bottom-up pass through the grammar anticipating items that would remain from the original structure and then proceed normally with Earley intersection. This bottom-up procedure does not perform predictions and its axioms are restricted to intersecting rules

$$\frac{}{[X \to \ \bullet \alpha, q, q] : u} \quad (q \in I) \wedge (X \underset{u}{\rightarrow} \in R) \qquad \text{axioms}$$

$$\frac{[X \to \alpha \bullet x\beta, q, s] : u}{[X \to \alpha x \bullet \beta, q, r] : u \otimes w(s, x, r)} \quad \langle s, x, r \rangle \in E \qquad \text{scan}$$

$$\frac{[X \to \alpha \bullet Y\beta, q, s] : u \ [Y \to \gamma \bullet, s, r] : v}{[X \to \alpha_s Y_r \bullet \beta, q, r] : u} \qquad \text{complete}$$

Figure 5.6: Bottom-up initialisation in incremental intersection

with only the initial state of the refining automaton.

Figure 5.6 shows the deductive program that describes this initialisation.[5] This procedure speeds up Earley intersection because it produces many of the items that Earley would produce without requiring an agenda. We simply iterate through the rules in the grammar in topological order of their left-hand side symbol (remember that we are talking about acyclic grammars) firing the operations shown in Figure 5.6. The items deduced by the program in Figure 5.6 initialise an agenda used to perform Earley intersection as shown in Figure 5.1. The complete items are made passive and the incomplete ones are active in this agenda. See in Figure 5.7 how at each iteration the incremental procedure performs about 3 times faster than the standard Earley intersection.

5.3.2 Selective intersection

In our application of OS* to SMT, each time a sample is rejected we collect information about n-gram contexts that are far too optimistic with respect to the true language model. This information is used to refine the proposal

[5]While this program might resemble the bottom-up intersection in (Dyer, 2010), they are actually different. Dyer's bottom-up procedure intersects an acyclic grammar with an acyclic automaton, there, axioms are instantiated for every state in the automaton (i.e. $\forall q \in Q$). Our procedure is used in the context of intersecting an acyclic grammar with a recursive automaton of the kind introduced in Section 4.2.3, and it is used as initialisation for the program in Figure 5.1.

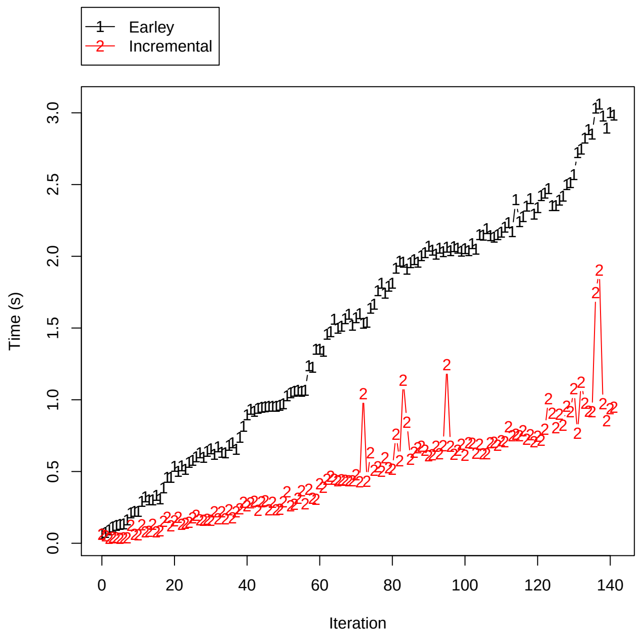
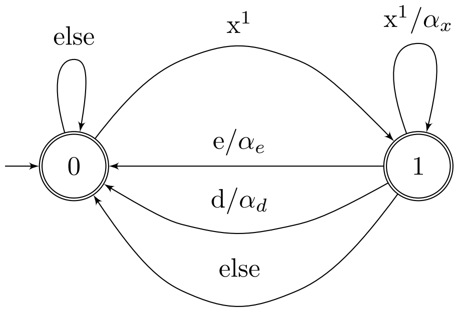


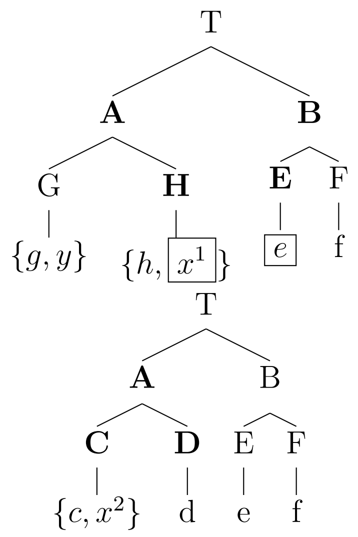
Figure 5.7: Incremental intersection

by incorporation of longer n-grams via intersection. However, these contexts might be compatible with many regions of the hypergraph, that is, with not only one, but many derivations. Simply observing the yield of rejected derivations does not directly reveal where in the hypergraph these contexts are likely to participate in high-scoring derivations, those likely to be sampled (or to be optimal).

Having in mind the grammar $q^{(0)}$ as illustrated earlier, consider a scenario in optimisation in which derivations in $q^{(0)}$ going through $\langle x, d \rangle$ are unlikely to be amongst the highest-scoring ones despite the optimistic assessments at the unigram level. That is, $T \overset{\delta_0^-}{\Rightarrow} xdef$, where $\delta_0^- = \{1, 4, 7, 13, 2, 9, 10\} \in R_0^*$ (see Figure 5.2a) scores poorly even in an optimistic assessment. In addition, suppose that we have gathered from $q^{(0)}$ convincing evidence to motivate a refinement that lowers bigrams of the kind $\langle x, \cdot \rangle$. For instance $T \overset{\delta_0^*}{\Rightarrow} gxef$, where $\delta_0^* = \{1, 3, 11, 6, 2, 9, 10\} \in R_0^*$, is the current optimum, and $\langle x, e \rangle$



(a) Selective refinement



(b) Selected interactions

Figure 5.8: Example of selective refinement $A^{(1)}$

realises the largest gap between $q^{(0)}$ and p, more specifically, between $A^{(0)}$ and the true LM distribution A. At this point one can refine $q^{(0)}$ with $A^{(0)}$ (Figure 5.3a) producing $q^{(1)}$ (Figure 5.5). While the refinement correctly down-weights all derivations compatible with $\langle x, \cdot \rangle$, including the optimum derivation δ_0^*, it also branches the grammar as to down-weight already unlikely derivations compatible with $\langle x, d \rangle$, such as δ_0^-.

The problem arises when one deals with real world grammars, which are considerably larger than those illustrations. Typically, there are many derivations compatible with a k-gram context α and incorporating it everywhere, as illustrated earlier, implies specialising the grammar as to make explicit everywhere whether or not a derivation's yield include the context α. This will affect all those derivations compatible with α irrespective of their possibly poor scores. Moreover, as far as we know — in optimisation mode — only one such derivation has been justified by a reject — the current optimum derivation. To proceed with the illustration, in which $\langle x, d \rangle$ is assumed unlikely even on optimistic assessment, there would be no need to branch the grammar producing the item $[C \rightarrow x \bullet, 0, 1]$ (Figure 5.4j). A lookahead at Figure 5.11 will give a better intuition of the purpose of *selectively* intersecting $q^{(0)}$. Note that fewer rules are produced and the packed forest is smaller than in the naïve case.

Suppose a scenario in which we have made each x in $q^{(0)}$ a different terminal, for instance, by annotating them with a sequential identifier that distinguishes different occurrences of x in the grammar. An example of that can be seen in Figure 5.8b, where different occurrences of x are annotated with a superscript (other terminals were left unnanotated because they only occur once). The updated set of terminals in $q^{(0)}$ is $\Sigma = \{c, d, e, f, g, h, x^1, x^2, y\}$ and the optimum derivation δ_0^* now yields $\langle g, x^1, e, f \rangle$. It is now possible to design a refinement that accounts for $\langle x^1, \cdot \rangle$ and not for $\langle x^2, \cdot \rangle$ (see Figure 5.8a) within the same framework already introduced.

Figure 5.9 illustrates this strategy which relies on Earley intersection and on the annotated version of the grammar. Figure 5.11 shows the resulting grammar, which is more compact than what was obtained before (Figure 5.5). While this seems efficient in making the effect of the refinement local, therefore avoiding an undesired combinatorial explosion in low-probability areas of the hypergraph, it is far too specific in the sense that very little is learnt from the rejected optimum in terms of overoptimistic n-grams. In a realistically sized grammar, there would exist many occurrences of x and even if most of them were located in low-probability regions, should at least 10% of them have high-probability, there would be more than just a few occurrences, and incorporating them one by one would be time-inefficient.

To better illustrate this inefficiency, consider the comparison shown in Figure 5.12. Curve 1 represents the intersection that incorporates any given k-gram context everywhere — let us call this “refining n-gram types” (because terminals of a certain surface “type” are handled altogether). Curve 2 represents a selective intersection in which terminals are annotated so that each occurrence becomes a unique symbol. Let us call this “refining n-gram instances” (because each terminal occurrence is treated as a different instance of a certain terminal base type).

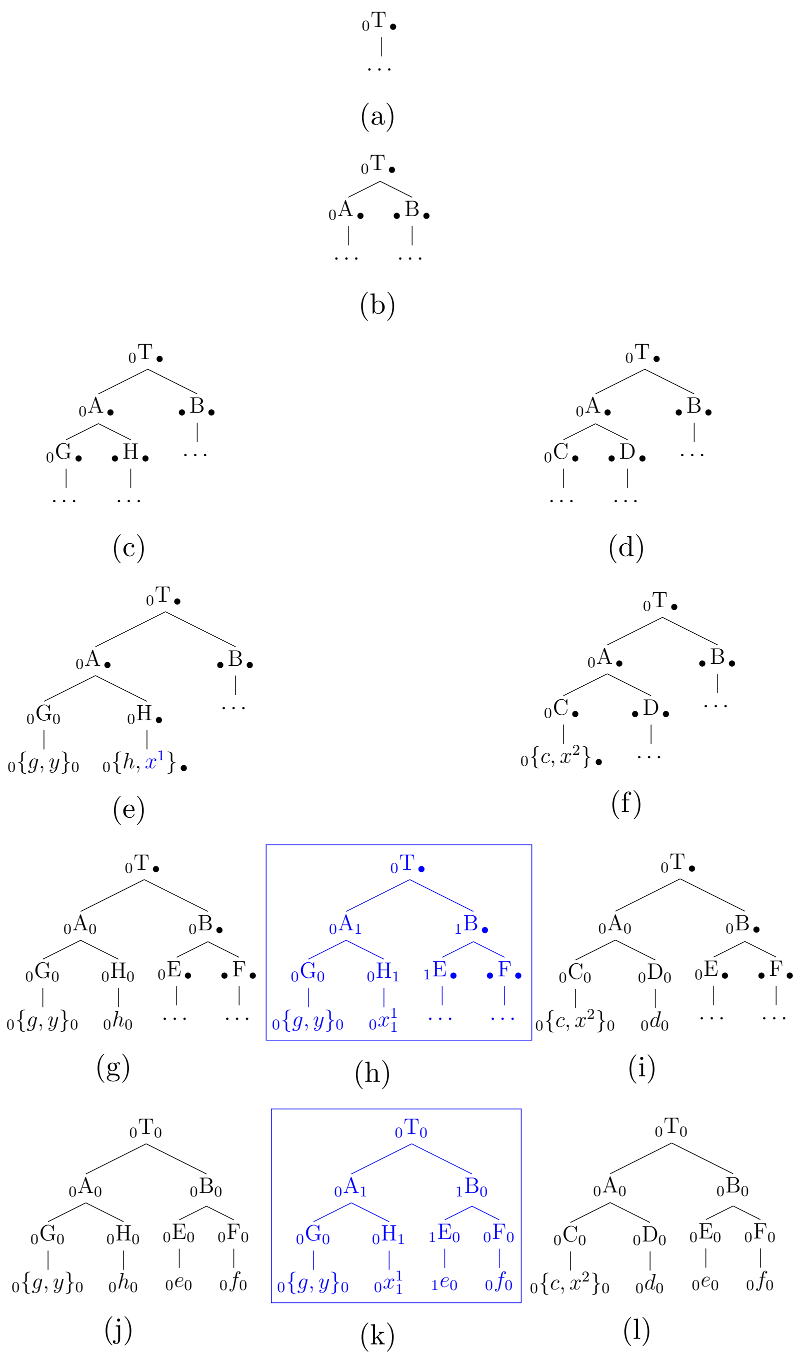
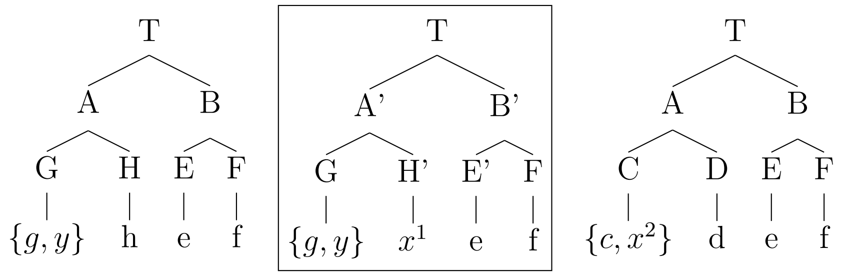


Figure 5.9: Intersecting the bigram $\langle x \bullet \rangle$ where it is more likely in $G(\mathbf{x})$

i	**Item**	**Rule**	**i**	**Item**	**Rule**
1	${}_0T_0 \xrightarrow[u_1]{} {}_0A_0\ {}_0B_0$	T $\xrightarrow[v_1]{}$ A B	7*	${}_0C_0 \xrightarrow[u_7 \otimes w(0,x^2,0)]{} x^2$	C $\xrightarrow[v_7]{} x^2$
+14	${}_0T_0 \xrightarrow[u_1]{} {}_0A_1\ {}_1B_0$	T $\xrightarrow[v_{14}]{}$ A' B'	8*	${}_0C_0 \xrightarrow[u_8 \otimes w(0,c,0)]{}$ c	C $\xrightarrow[v_8]{}$ c
2	${}_0B_0 \xrightarrow[u_2]{} {}_0E_0\ {}_0F_0$	B $\xrightarrow[v_2]{}$ E F	9*	${}_0E_0 \xrightarrow[u_9 \otimes w(0,e,0)]{}$ e	E $\xrightarrow[v_9]{}$ e
+15	${}_1B_0 \xrightarrow[u_2]{} {}_1E_0\ {}_0F_0$	B' $\xrightarrow[v_{15}]{}$ E' F	+17	${}_1E_0 \xrightarrow[u_9 \otimes w(1,e,0)]{}$ e	E' $\xrightarrow[v_{17}]{}$ e
3	${}_0A_0 \xrightarrow[u_3]{} {}_0G_0\ {}_0H_0$	A $\xrightarrow[v_3]{}$ G H	10*	${}_0F_0 \xrightarrow[u_{10} \otimes w(0,f,0)]{}$ f	F $\xrightarrow[v_{10}]{}$ f
+16	${}_0A_1 \xrightarrow[u_3]{} {}_0G_0\ {}_0H_1$	A' $\xrightarrow[v_{16}]{}$ G H'	11*	${}_0G_0 \xrightarrow[u_{11} \otimes w(0,g,0)]{}$ g	G $\xrightarrow[v_{11}]{}$ g
4	${}_0A_0 \xrightarrow[u_4]{} {}_0C_0\ {}_0D_0$	A $\xrightarrow[v_4]{}$ C D	12*	${}_0G_0 \xrightarrow[u_{12} \otimes w(0,y,0)]{}$ y	G $\xrightarrow[v_{12}]{}$ y
5*	${}_0H_0 \xrightarrow[u_6 \otimes w(0,h,0)]{}$ h	H $\xrightarrow[v_6]{}$ h	13*	${}_0D_0 \xrightarrow[u_{13} \otimes w(0,d,0)]{}$ d	D $\xrightarrow[v_{13}]{}$ d
6*	${}_0H_1 \xrightarrow[u_5 \otimes w(0,x^1,1)]{} x^1$	H' $\xrightarrow[v_5]{} x^1$			

Figure 5.10: Rules



(a) Forest

Figure 5.11: Selectively intersected grammar $q^{(1)}$

Each curve in Figure 5.12 shows the time necessary to perform the intersection that accounts for some k-gram context in a optimisation task with OS* and an upperbound on a 4-gram language model for a short input sentence containing 7 words. Observe that, when refining on types (curve 1), the true optimum is found after just about 30 iterations. This means that less than 30 different k-gram contexts of variable order need to be accounted for in order to obtain a solution with a certificate of optimality. Curve 1 is steep, showing that the grammar grows considerably after each intersection making the next intersection slower. On the other hand, when refining on instances (curve 2), faster intersections are observed. This happens because at each iteration only a very specific instance of a certain k-gram context is incorporated — the instance that participates in the rejected optimum — therefore, only derivations compatible with that given instance will be specialised. While this extreme form of selective intersection enables a fine control over the growth of the grammar, there is very little generalisation from the rejected optimum and not many derivations have their upperbound scores lowered. Ultimately, such conservative refinements lead to many more iterations before optimality can be proven (about 140 iterations in the example).

To add to the illustration, consider the terminal type *the*. The initial proposal associated to Figure 5.12 contained 108 instances of *the*. Intersecting those instances naïvely branches the grammar for all the potentially 108 rules involving *the*. Intersecting those instances carefully, one by one as they show up in rejected derivations, requires accounting for only 16 of them. For all 1-word contexts that are incorporated before reaching the true optimum in that illustration, refining on types incorporates 62% of all terminal instances of the initial hypergraph, while refining on instances only touches 7% of them. Finally, it is important to stress that the same n-gram types are incorporated by both strategies, however, in the more conservative case (curve 2), fewer instances of those types are refined.

While this observation reveals more about the nuances of the problem, it does not offer a concrete solution. The total decoding time, given by the "area" below the curves (or sum of the bars) in Figure 5.12, is still much higher in the selective case. Now consider the strategy we have just introduced, which incorporates n-grams "instance-by-instance", in sampling mode. At each iteration, sampling can offer a much more insightful view of the distribution by simply drawing a "large" batch of samples (say a few hundreds) from the proposal, instead of a single derivation such as it happens in optimisation. In this batch of samples one will typically find many occurrences of a certain terminal type, but only a few instances will be

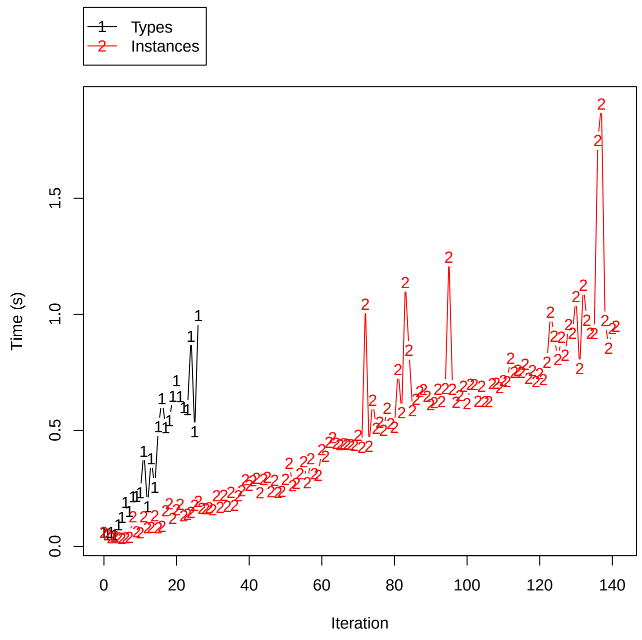


Figure 5.12: Refining n-gram types vs. refining n-gram instances

responsible for most of the evidence supporting a refinement on that type. In the illustration we have just presented, we are likely to identify most of the 16 “popular” (or high-scoring) instances of the terminal type *the* in the first round of sampling.

This motivating argument reveals the goal, as well as the nature, of what we call selective intersection. Selective intersection aims at incorporating context in high-probability regions of the hypergraph, and it is connected to how much probability mass the grammar allocates to strings under different subderivations. This intuitive formulation can be formalised in terms of marginal edge probabilities, that is, the total probability of sampling a certain edge out of the hypergraph. Rephrasing it further to make it clearer, the chance of a rule participating in a derivation sampled at random from the distribution. Formally this notion of “edge marginal”, or “rule marginal”, changes depending on the choice of semiring. While in a sum-times semiring it represents the sum of the probabilities of all derivations that utilises the given rule, in a max-times semiring it represents the score of the best derivation that utilises the rule.

5.3.2.1 Strategies of selective intersection

Let us formalise a more general strategy for selective intersection. Given a grammar $q^{(t)} = \langle \Sigma, V, \langle S, \sigma \rangle, \langle R, \nu \rangle \rangle$, let us start by recollecting from Section 3.1.1 the functions: $\mathsf{I} : V \to \mathbb{K}$, which takes a nonterminal (also called "node") in V and returns its *inside weight*; $\mathsf{O} : V \to K$, which takes a nonterminal and returns its *outside weight*; and $\mathsf{M} : R \to K$, which takes a rule in R and returns its *marginal weight*. By abuse of language let us also write $\mathsf{M}[z]$, for $z \in \Sigma$, when referring to $\mathsf{O}[z]$, that is, the outside weight of a terminal.[6] Moreover, let $\alpha \in \Sigma^k$ be a k-gram context, $z \in \Sigma$ be the last word of the n-gram αz, and z^i be a specific instances of terminal z.

At first, let us focus on bigrams αz, where $\alpha \in \Sigma$ is a 1-word context. Often we will say "incorporating α" (or "refining on α") meaning the intersection of the current grammar with an automaton that reweights all 1-word continuations of α. Suppose we gather evidence to support the refinement on a certain unigram type $\alpha = x$ such that no instance of this type has been incorporate before. We can use the current grammar to split the set of instances of x into two sets. The first set is made of the high-scoring instances of x, lets call it H_x for *high*. The second set is made of the remaining (lower-scoring) instances of x, let us call it L_x for *low*. Let us assume that by high-scoring we mean the top-scoring instances that represent a minimum portion of the total mass assigned to the type (say 10% of the mass). This split can be performed by sorting instances of x according to a normalised view of $\mathsf{M}[x]$, denoted by $\bar{\mathsf{M}}[x]$, and selecting instances x^i in descending or-

[6]Note that to every terminal z one can associate a single pre-terminal rule r such that $\langle z \rangle \xrightarrow[\bar{1}]{} z$, where $\langle z \rangle$ represents a unique nonterminal made up from z, and the marginal weight of r, $\mathsf{M}[r] = \mathsf{O}[z] \otimes \mathsf{I}[z] = \mathsf{O}[z] \otimes \bar{1} = \mathsf{O}[z]$, is equal to the outside weight of z.

der of $\mathsf{M}[x^i]$, such that the total mass in H_x exceeds a threshold. That is, by selecting H_x such that $\sum_{x^i \in H_x} \bar{\mathsf{M}}[x^i] \geq th$, where normalisation happens with respect to the total mass assigned to the type, i.e. $\bar{\mathsf{M}}[x^i] = \frac{\mathsf{M}[x^i]}{\sum_j \mathsf{M}[x^j]}$. Refinements are then dynamically designed to specialise the proposal only with respect to the instances in H_x, leaving the instances in L_x at a lower-order representation.

Following the discussion in Section 4.2.4, the coarse-to-fine search performed by OS* requires keeping track of what k-gram contexts have already been incorporated to the implicit upperbound LM distribution A_0^t. In the case where instances are dealt with in groups, this bookkeeping might become non-trivial. Suppose at some point we refine the high-scoring instances of a terminal type x, say $H_x = \{x^1, x^2, x^3\}$ and $L_x = \{x^4, \ldots, x^m\}$. Nothing prevents OS* from requiring the incorporation of some $x^l \in L_x$ at some later point, after all, selective intersection is not synonym of pruning, but rather a way of anticipating computations that are likely to happen anyway and delaying computations of which we are unsure will happen. This instance x^l may represent some important computation that was overlooked the first time the split H_x and L_x was estimated, either because the threshold was not inclusive enough or because the instance was not likely at such an early stage of the search, redeeming itself later on (what could never happen if the instance had fallen off a beam in cube-pruning). For these redeemed instances, such as x^l, we opt to resort to the simpler and more conservative strategy of incorporating the specific instance solely.[7] In terms of bookkeeping, the set of high-scoring instances of x is updated $H_x \leftarrow H_x \cup \{x^l\}$.

Let us now turn to higher order contexts, starting with αz, where $\alpha = yx$ is a bigram. Consider $I_y \subseteq (H_y \cup L_y)$ and $I_x \subseteq H_x$, where the instances H_x have already been incorporated as 1-word contexts. Assume the instance

[7]Note that we could incorporate additional instances, not yet in H_x, in one go, but it turns out it is efficient enough to treat the remaining instances as exceptions to the rule and incorporate them one by one as they redeem themselves from L_x.

$y^k x^h$ a 2-word context to be incorporated, where $(y^k, x^h) \in I_y \times I_x$. Let us also define the set of high-probability instances of the type yx as $H_{yx} = \{\langle y^j, x^i \rangle : y^j \in I_y \wedge x^i \in I_x\} \subseteq H_y \times H_x$ and $y^k x^h \in H_{yx}$. On one extreme, when $I_y = \{y^k\}$ and $I_x = \{x^h\}$, we could incorporate $y^k x^h$ alone, and leave other instances of yx to some later point when they are proven necessary. On the other extreme, when $I_y = (H_y \cup L_y)$ and $I_x = H_x$, we could incorporate all extensions (by any instance of y) of instances in H_x.[8] However, when y and x interact, a smaller subset of $(H_y \cup L_y) \times H_x$ typically represents most of the mass associated to yx.[9] A compromise solution is to select $I_y \subseteq (H_y \cup L_y)$ and $I_x \subseteq H_x$, so that H_{yx} holds at least a minimum portion of the total mass in $H_y \times H_x$. For instance, Figure 5.13 illustrates how one can choose the subset of I_y and I_x. Suppose each cell contains the product $M(y^{j^{th}}) \otimes M(x^{i^{th}})$ normalised with respect to all (y^j, x^i) pairs in $(H_y \cup L_y) \times H_x$. Moreover, y's and x's are sorted in descending order. In the figure, darker cells represent more probability mass, and the four cells at the top-left corner hold more than 40% of the total mass. When at a later point another bigram context $\langle y^j, x^i \rangle \notin H_{yx}$, and $x^i \in H_x$, needs to be accounted for, we proceed by simply adding that specific instance.

For context $\alpha \in \Sigma^k$, where $k > 2$, everything is the same. If no instance of α has ever been incorporated, we proceed by choosing the set H_α as before. Otherwise, we incorporate the specific instance solely. As the length of the context increases, a computation of the kind shown in Figure 5.13 becomes expensive due to normalisation. A different criterion is to make use of a parameter β that restricts how far from the best interaction one can be. This parameter works like a beam, but rather than establishing a limit beyond which interactions get pruned, it establishes how much computation will be

[8]Note that choosing $H_{yx} = (H_y \cup L_y) \times (H_x \cup L_x)$ would not make sense, since that set includes instances $y^j x^i$, where $x^i \in L_x$, which have not yet been incorporated at a lower level.

[9]Typically one can limit I_y to a subset of H_y and still anticipate most of the computation that would be required to lower the upperbound sufficiently.

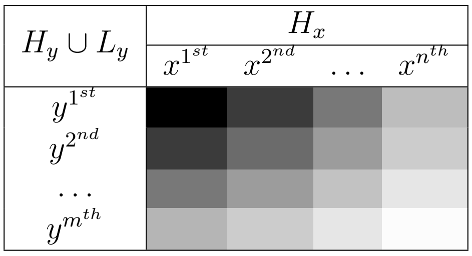


Figure 5.13: Interaction between instances in a bigram

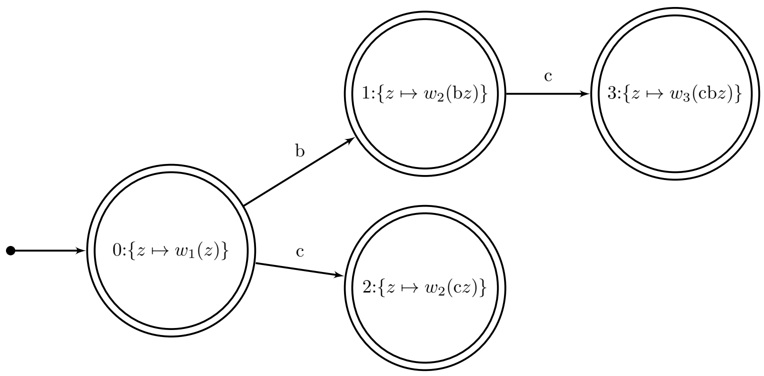
anticipated, leaving the remaining to some later time when explicit evidence becomes available.

5.3.2.2 Estimating edge marginals with sampling

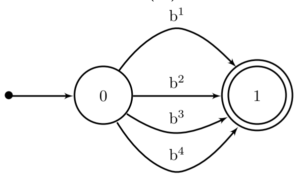
Finally, in terms of time-efficiency, computing $\mathsf{M}[z]$ at each iteration requires running the Inside-Outside algorithm (Section 3.1.1), which is $O(|G|^2)$. Alternatively, $\mathsf{M}[z]$ can be kept from iteration to iteration and only recomputed when eventually an k-gram instance of a k-gram type never refined before falls in the low-probability set. This might still prove inefficient, thus sampling can also be used to quickly obtain a summary of the distribution, from which $\mathsf{M}[z]$ can be quickly estimated. Moreover this estimation is unbiased given that samples are drawn independently from the proposal.

5.3.2.3 Bookkeeping

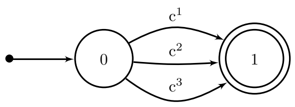
Bookkeeping now is performed with respect to two dimensions, the types and the instances. A trie $T = \langle \Sigma, Q, I = \{q_0\}, F = Q, E \rangle$ is used to bookkeep k-grams over the vocabulary Σ of terminal types. Just like it was discussed in Section 4.2.4, there is a one-to-one correspondence between each state in the trie and a k-gram context $\alpha \in \Sigma^k$. The difference now is that to which state $q \in Q$, in addition to the function $\{z \mapsto w_k(\alpha z)\}$, with $z \in \Sigma$, we also associate a non-deterministic automaton $I_\alpha = \langle \Sigma'_\alpha \subseteq \Sigma', Q', I', F', E' \rangle$ that compactly encodes the subset of instances of α that have already been incorporated. The set Σ' is that of all terminal instances in the proposal


0:$\{z \mapsto w_1(z)\}$
b
c
1:$\{z \mapsto w_2(\mathrm{b}z)\}$
2:$\{z \mapsto w_2(\mathrm{c}z)\}$
c
3:$\{z \mapsto w_3(\mathrm{cb}z)\}$

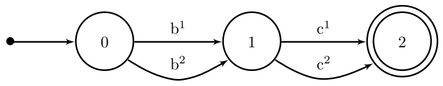
(a) Trie used to bookkeep n-gram types



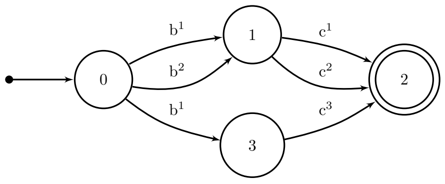
(b) Instances of b



(c) Instances of c



(d) Instances of bc



(e) Updated instances of bc

Figure 5.14: Bookkeeping in selective intersection

hypergraph, and Σ'_α is a subset of Σ' made of instances of the types participating in the context α only. Choosing I_α to be non-deterministic is a convenience that makes it simpler to maintain the information it encodes.

To illustrate how this structure is maintained, suppose an initial proposal $q^{(0)}(\mathbf{d}) = G(\mathbf{x}) \cap A^{(0)}$, where $A^{(0)}$ incorporates $w_1(u[z^i])$ for every instance z^i based on a type $z \in \Sigma$.[10] At this point, the trie used for bookkeeping contains only state 0 shown in Figure 5.14a. Suppose a derivation optimised from $q^{(0)}$ yields $c^1b^1a^1$. The bigram candidates for refinement are c^1b^1 and b^1a^1, because instances of cb or ba have never been accounted for. Suppose $\frac{w_2(ba)}{w_1(a)} < \frac{w_2(cb)}{w_1(b)}$,

[10]Note that although in $q^{(0)}$ terminals are instances, the upperbound weight is a function of terminal types only, that is, the LM only cares about unannotated strings.

that is, b^1a^1 realises the largest gap between $q^{(0)}$ and p. Therefore we proceed by choosing H_b the set of high-scoring instances of b making sure $b^1 \in H_b$.[11] Assume $H_b = \{b^1, b^2, b^3, b^4\}$ is chosen, then a refinement $A^{(1)}$ is designed to reweight 1-word continuations of instances in H_b, and $q^{(1)}$ is obtained by the intersection $q^{(0)} \cap A^{(1)}$. The trie is updated by creation of state 1 and the transition $\langle 0, b, 1\rangle$ which memorises that the type b has been incorporated. State 1 stores both the function $\{z \mapsto w_2(\mathrm{b}z)\}$, for $z \in \Sigma$, and a pointer to the automaton shown in Figure 5.14b, which memorises that only instances in H_b have been intersected.

Consider the optimum in $q^{(1)}$ yields $b^1c^1a^1$; the bigrams are b^1c^1 and c^1a^1, but b^1c^1 is not a candidate. Note how the (reversed) prefix b^1 has been memorised by the trie. We first find the state associated to the type b in Figure 5.14a, i.e., 1. Then, we check if the automaton associated with it recognises b^1. From Figure 5.14b, that would be the case. Thus, c^1a^1 is the only candidate, and bigrams starting in instances of c have never been accounted for. Therefore we proceed by deciding on H_c such that $c^1 \in H_c$, designing the automaton $A^{(2)}$, obtaining $q^{(2)}$ by intersection and updating the trie. The trie now contains state 2 and a transition $\langle 0, c, 2\rangle$. State 2 also contains an automaton, shown in Figure 5.14c, to memorise which instances of c have been selected in H_c, and the function $w_2(\mathrm{c}z)$.

Now, suppose the optimum in $q^{(2)}$ yields $c^1b^1a^1$ again; the next available refinement is to account for continuations of c^1b^1. Note that b^1 has been memorised, and no instance of cb has ever been incorporated. Thus, we decide on H_{cb}, such that $c^1b^1 \in H_{cb}$, designs the automaton $A^{(3)}$, obtains the proposal $q^{(3)}$ by intersection, and updates the trie. The trie is updated with state 3, which is associated to the type cb, and the transition $\langle 1, c, 3\rangle$. State 3 stores $w_3(\mathrm{cb}z)$ and the automaton shown in Figure 5.14d, which

[11]Typically $b^i \in H_b$, if this is not the case in some odd scenario, it is easy to make sure $b^i \in H_b$ by choosing H_b conveniently.

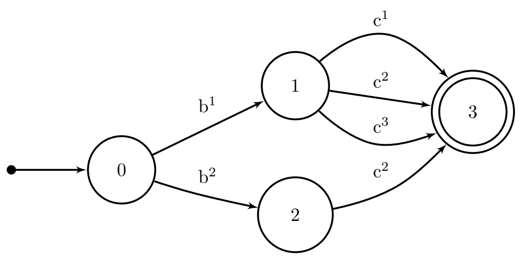


Figure 5.15: Deterministic version of the automaton "instances of bc"

memorises the instances in H_{cb}. Note how the automaton efficiently encodes the cross-product $\{b^1, b^2\} \times \{c^1, c^2\}$.

Finally, suppose that at some later time t, the optimum in $q^{(t)}$ yields $c^3 b^1 a^1$. The prefix we are looking for is $c^3 b^1$, its reverse is $b^1 c^3$ and the trie does accept the type bc, however, from Figure 5.14d we can see that $b^1 c^3$ is not accepted. Thus, we assume that $c^3 b^1$ redeemed itself from L_{cb} and design a refinement $A^{(t+1)}$ that reweights 1-word continuations of that specific instance obtaining $q^{(t+1)}$ by intersection. In this case, the automaton in Figure 5.14d is updated to the one shown in Figure 5.14e. Note that state 3 is added so that $b^1 c^3$ is accepted, but $b^2 c^3$ is not. Here the non-determinism of this automaton makes it easy to memorise different instances of cb that share some, but not all, suffixes. That is, $H_b \supset H_b \cap I_b \neq \emptyset$, for $I_b = \{b^i : \exists(\cdot, b^i) \in H_{cb}\}$.

5.4 Conditioning context

It is sometimes useful to know the set of context histories that may precede a certain word. For instance, we can use this information about conditioning contexts to tighten the upperbound on the language model distribution. In Section 4.2.2.2, we discussed how a 1-word conditioning context can be used to perform such tightening. In fact, longer histories could be used for this purpose. However, the algorithm we will discuss for finding the set of active n-grams compatible with a given grammar runs in time proportional to

$$\frac{}{[S' \to \bullet X, \langle \mathrm{BOS}_1, \ldots, \mathrm{BOS}_k \rangle \star \langle \mathrm{BOS}_1, \ldots, \mathrm{BOS}_k \rangle]} \; X \in S \qquad \text{AXIOMS}$$

$$[S' \to X \bullet, \mathbf{y}_1 \star \mathbf{y}_2] \;\; X \in S \qquad \text{GOALS}$$

$$\frac{[X \to \alpha \bullet Y\beta, \mathbf{y}_1 \star \mathbf{y}_2]}{[Y \to \bullet \gamma, \mathbf{y}_2 \star \mathbf{y}_2]} \; Y \to \gamma \in R \qquad \text{PREDICT}$$

$$\frac{[X \to \alpha \bullet z\beta, \mathbf{y}_1 \star \mathbf{y}_2]}{[X \to \alpha z \bullet \beta, \mathbf{y}_1 \star \langle \mathbf{y}_2 z \rangle_k]} \qquad \text{SCAN}$$

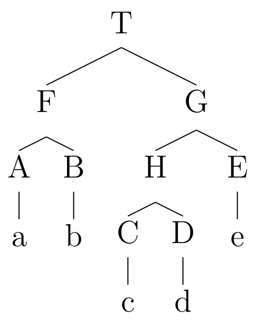
$$\frac{[X \to \alpha \bullet Y\beta, \mathbf{y}_1 \star \mathbf{y}_2] \; [Y \to \gamma \bullet, \mathbf{y}_2 \star \mathbf{y}_3]}{[X \to \alpha Y \bullet \beta, \mathbf{y}_1 \star \mathbf{y}_3]} \; X \neq S' \qquad \text{COMPLETE}$$

$$\frac{[S' \to \bullet X, \mathbf{y}_1 \star \mathbf{y}_2] \; [X \to \gamma \bullet, \mathbf{y}_2 \star \mathbf{y}_3]}{[S' \to X \bullet, \mathbf{y}_1 \star \langle \mathbf{y}_3 \, \mathrm{EOS} \rangle_k]} \; X \in S \qquad \text{ACCEPT}$$

Figure 5.16: Logic program that enumerates the n-grams in $G(\mathbf{x})$.

$|\Sigma|^{2(n-1)}$, where Σ is the vocabulary of terminals in the grammar. Moreover, $|\Sigma|$ is typically proportional to the length I of the input sentence.

The program in Figure 5.16 enumerates the n-grams in $G(\mathbf{x})$. Its input is a grammar $G(\mathbf{x}) = \langle \Sigma, V, \langle S, \sigma \rangle, \langle R, \nu \rangle \rangle$ and the parameter $k = n - 1$. An item of the program has the form $[X \to \alpha \bullet \beta, \mathbf{y}_1 \star \mathbf{y}_2]$. The first part $X \to \alpha \bullet \beta$ is a dotted rule. It means that we are trying to match under X a span $\alpha\beta$, where $\alpha \in (V \cup \Sigma)^*$ has already been recognised and $\beta \in (V \cup \Sigma)^*$ is yet to be recognised. The second part $\mathbf{y}_1 \star \mathbf{y}_2$ is a pair of terminal strings, each of which contains exactly $k = n - 1$ words. The string $\mathbf{y}_1$ (or *history*) represents the k words that precede the rule's span $\alpha\beta$. The string $\mathbf{y}_2$ (or *suffix*) represents the k words that precede the dot. The star ($\star$) simply illustrates the fact that part of the string from the first word in $\mathbf{y}_1$ to the last word in $\mathbf{y}_2$ might have been elided. An item is *complete* if its dot has reached the end of the rule, otherwise the item is *incomplete*. Finally, the operator $\langle \mathbf{y} z \rangle_k$ concatenates the string $\mathbf{y}$ with the terminal z and returns the



Item	**Fire**	**3-gram**
$[G \to \bullet HE, ab \star ab]$	PREDICT	
$[H \to \bullet CD, ab \star ab]$	PREDICT	
$[C \to \bullet c, ab \star ab]$	SCAN	abc
$[C \to c \bullet, ab \star bc]$	COMPLETE	
$[H \to C \bullet D, ab \star bc]$	PREDICT	
$[D \to \bullet d, bc \star bc]$	SCAN	bcd
$[D \to d \bullet, bc \star cd]$	COMPLETE	
$[H \to CD \bullet, ab \star cd]$	COMPLETE	
$[G \to H \bullet E, ab \star cd]$	PREDICT	
$[E \to \bullet e, cd \star cd]$	SCAN	cde
$[E \to e \bullet, cd \star de]$	COMPLETE	
$[G \to HE \bullet, ab \star de]$	COMPLETE	

Figure 5.17: Role of *history* and *suffix* in enumerating 3-grams

k last words in the resulting sequence.

The program starts (AXIOMS) by initialising dotted rules of the kind $S' \to \bullet X$ where S' is an artificial root symbol (not in V) and $X \in S$ is a start symbol of $G(\mathbf{x})$. These items are prefixed by a sequence of k begin-of-sentence symbols. Note that at this point the dot is before the beginning of the rule's right-hand side, therefore, the history and the suffix are the same. The GOAL of the program is to prove complete items rooted by a start symbol. In PREDICT, new items are created by matching rules whose left-hand side is the nonterminal Y after the dot. Observe that the consequent inherits the string $\mathbf{y}_2$ since that is the one that precedes the dot in the antecedent. Again, because the dot precedes the beginning of the rule's right-hand side, the history and the suffix are identical in the consequent. In SCAN, the dot is moved forward over a terminal. In the consequent the suffix is updated to incorporate the terminal just scanned. Observe that scanning does not change the history $\mathbf{y}_1$ of the item. COMPLETE combines two items: the first is an incomplete item whose dot precedes a nonterminal Y; the second is a complete item whose left-hand side is also Y and whose history $\mathbf{y}_2$ matches the first item's suffix. In relation to the first item, the deduction moves its dot forward over Y, it retains the original history $\mathbf{y}_1$ and it updates the suffix to $\mathbf{y}_3$. Finally, in ACCEPT, we introduce the end-of-sentence symbol producing a goal item.

Figure 5.17 illustrates how 3-grams are enumerated using the program. Let us focus on the subderivation rooted at G. The bigram ab precedes the span covered by G in the yield of the derivation shown. The item that holds this information is denoted by $[G \to \bullet HE, ab \star ab]$. Note that because the dot is at the beginning of the rule, the history and the suffix are the same. We proceed firing operations aiming at moving the dot forward and producing complete items. As the dot progresses within a rule, the history is preserved and the suffix is updated. When we perform prediction the history is updated. The figure also illustrates how SCAN enumerates 3-grams. It has access to a bigram suffix which precedes the dot and the terminal just after the dot.

The program is very similar to Earley intersection. In fact, except for the weights, it is implicitly equivalent to intersecting the translation grammar with an automaton that represents the complete n-gram language model.[12] Observe that, each time a terminal is scanned, the program has access to a k-word-long conditioning context and the terminal that succeeds it. Therefore, we can easily design SCAN to memorise the set of n-grams compatible with $G(\mathbf{x})$. Following the notation in Figure 5.16, $\langle \mathbf{y}_2 z \rangle$ in SCAN is one such n-gram. The number of items in the program is bounded by $r|R||\Sigma|^{2(n-1)}$, where r is the length of the longest right-hand side plus one (i.e., the maximum number of positions for the dot), $|R|$ is the size of the set of rules and $|\Sigma|$ is the size of the vocabulary of terminals in $G(\mathbf{x})$. Because we need to track two strings of $k = n - 1$ terminals each, the complexity is proportional to $|\Sigma|^{2(n-1)}$. This program is prohibitively slow for $n > 2$, and even for $n = 2$

[12]It is straightforward to incorporate the grammar and the LM weights. Axioms are weighted by $\bar{1}$; PREDICT weights the consequent by the rule weight; SCAN weights the consequent by the $\otimes$-product between the weight of the antecedent and $p_{lm}(z|\mathbf{y}_2)$; COMPLETE weights the consequent by the weight of the first antecedent; and ACCEPT incorporates $p_{lm}(\text{EOS}|\mathbf{y}_3)$.

(bigrams) it is only convenient for very short sentences.

Enumerating the n-grams that might participate in the yield of derivations in $G(\mathbf{x})$ is just as computationally expensive as performing a full intersection. Therefore, for the purpose of tightening LM upperbounds, we enumerate only the bigrams in $G(\mathbf{x})$. To perform this computation quickly, we relax the program in Figure 5.16 with respect to the string that represents the history of the item. Observe in Figure 5.16 that the history constrains item combination in COMPLETE. Without such constraint, we might create items that do not really participate in any derivation. Therefore some bigrams that are not strictly compatible with $G(\mathbf{x})$ will be memorised by SCAN. Except for the axioms of the program, the *history* is always set by the PREDICT rule. Since we are disregarding the history, there is no need to perform prediction and we can proceed with a bottom-up program that visits the nodes in topological order without the need to manage a complex agenda of items.

The program in Figure 5.18 illustrates a fast procedure to enumerate this superset of the set of bigrams in $G(\mathbf{x})$. The axioms of the program iterate over the set of rules of the grammar creating two types of items. Items of the form $[X, l, r]$ summarise a rule in the grammar, where X is the left-hand side nonterminal, $l \in (V \cup \Sigma)$ is the first symbol of the rule's right-hand side, and $r \in (V \cup \Sigma)$ is the last symbol of the rule's right-hand side. Note that l and r are the same if the right-hand side has a single symbol. Items of the form $\langle l, r\rangle \in (V \cup \Sigma)^2$ represent bigrams of terminal and nonterminal symbols. These items represent all sequences of two adjacent symbols in the right-hand side of the rules of the grammar. The GOAL of the program is to prove pairs of terminal symbols. The INFERENCE RULES expand items of the kind $\langle l, r\rangle$ where at least one of the symbols is a nonterminal. These rules can be exhaustively applied in a single pass over the nonterminal vocabulary in topological order. Expanding items take up to $|V|$

Axioms

$$\frac{}{[X, \alpha_1, \alpha_n]} \quad X \rightarrow \alpha_1 \ldots \alpha_n$$

$$\frac{}{\langle \alpha_i, \alpha_{i+1} \rangle} \quad X \rightarrow \alpha_1 \ldots \alpha_n \wedge n > 1 \wedge i \in [1, n-1]$$

Goals

$$\langle z_1, z_2 \rangle \in \Sigma^2$$

Inference rules

$$\frac{\langle X, r \rangle \; [X, l_x, r_x]}{\langle r_x, r \rangle} \quad X \in V$$

$$\frac{\langle l, X \rangle \; [X, l_x, r_x]}{\langle l, l_x \rangle} \quad X \in V$$

Figure 5.18: Logic program that enumerates a tight superset of the set of bigrams in $G(\mathbf{x})$.

deductions for each nonterminal. Therefore the program in Figure 5.18 runs in time $O(|V|^2)$. Producing this tight superset of the set of bigrams in $G(\mathbf{x})$ is reasonably fast and allows us to further tighten the initial upperbound on the LM distribution.

5.5 Experiments

For the experiments reported in this section we used the German-English portion of the 6th version of the Europarl collection (Koehn, 2005). In all experiments, we used the Moses toolkit (Koehn et al., 2007) to extract a wSCFG following Chiang (2005).[13] Language models were trained by *lmplz* (Heafield et al., 2013) using the English monolingual data made available by the WMT (Callison-Burch et al., 2012). That is, *Europarl*, *newscommen-*

[13]We extracted grammars containing at most two nonterminals on the right-hand side and at most 10 target productions for a given source production.

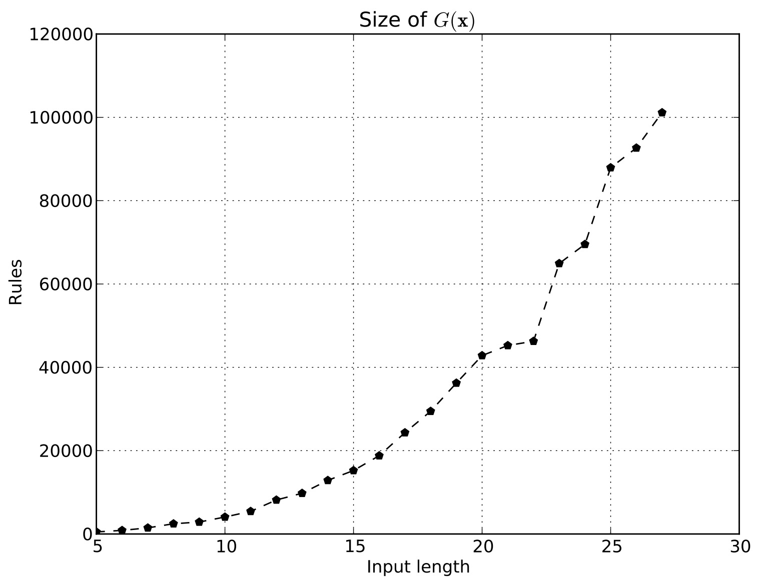


Figure 5.19: Number of rules in $G(\mathbf{x})$ as a function of sentence length

taries, *news-2012* and *commoncrawl*. The models extracted from different monolingual corpora were combined via interpolation using *newstest2010* as the development set. In all experiments, parameter estimation was performed via MERT on the *newstest2010* development set using single reference BLEU as the oracle. Finally, from the *newstest2011* test set, we selected sentences randomly according to their length. We sampled 20 examples for each class of length from 1 to 30 words.

Let us start by inspecting some properties of the initial grammar $G(\mathbf{x}) = (\mathcal{X} \circ \mathcal{G}) \downarrow$. Figure 5.19 shows the average number of rules in the initial grammar as a function of the sentence length. Observe how the growth appears to be cubic, which is a consequence of having up to two nonterminals on the right-hand side of a rule. Figure 5.20 shows the average length of the longest rule (in terms of its right-hand side) in the grammar as a function of the sentence length. We see that even short sentences tend to include long rules. However, recall that the maximum length of a rule is bounded by a

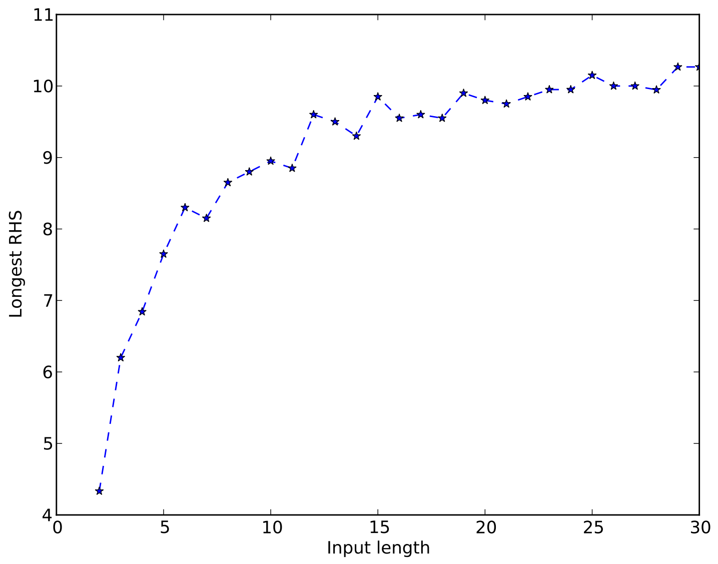


Figure 5.20: Longest rule in $G(\mathbf{x})$ as a function of sentence length

constant due to the grammar extraction procedure (Chiang, 2005). In the context of intersecting a grammar with an automaton, the quantity r, which is defined as the length of the longest rule plus one, represents the longest sequence of wFSA states that annotate a rule in the grammar.

Figure 5.21 shows the average number of target words (terminals) as a function of the sentence length. The terminals of $G(\mathbf{x})$ make the active vocabulary of target words. That is, the initial automaton $A^{(0)}$, which represents an optimistic unigram version of the full LM, contains a single state and a looping transition for each word/terminal/unigram in that set. Sometimes we refer to the set of terminals as the set of unigrams that are compatible with $G(\mathbf{x})$, meaning the set of target words that can participate in derivations from $G(\mathbf{x})$. Note how its growth is close to linear, which is due to the fact that the number of synchronous rules per source segment is bounded.

Figure 5.22 shows the average number of unique bigrams in $G(\mathbf{x})$. The starred line is the size of the set of bigrams in $G(\mathbf{x})$ computed with the fast

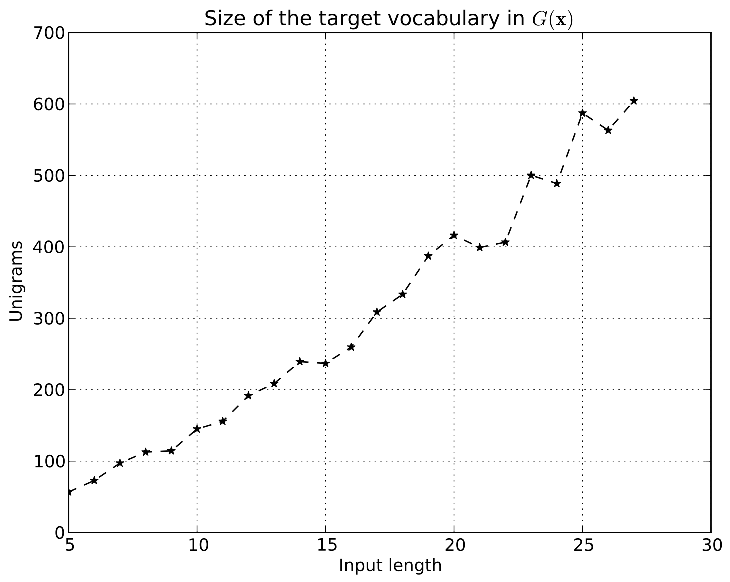


Figure 5.21: Target unigrams in $G(\mathbf{x})$ as a function of sentence length

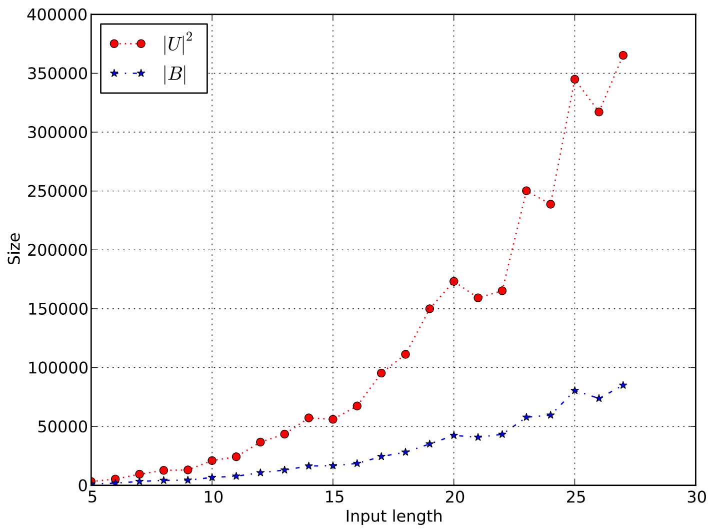


Figure 5.22: Target bigrams compatible with $G(\mathbf{x})$

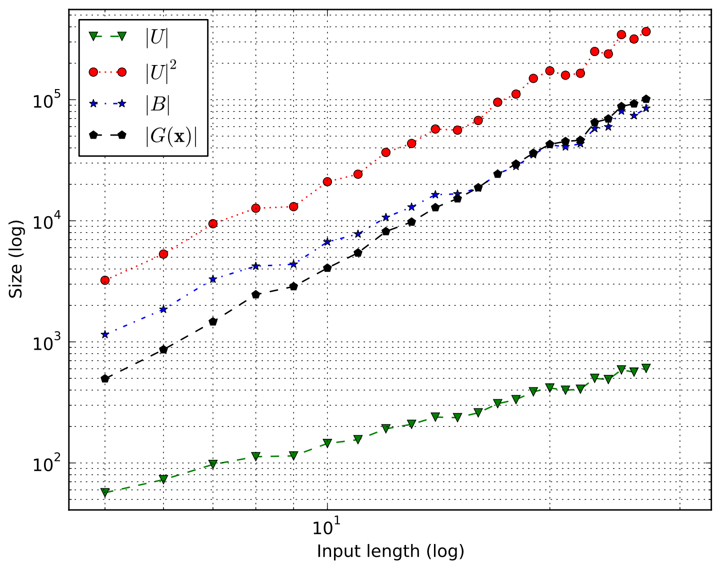


Figure 5.23: Size of the target vocabulary and of the set of rules in $G(\mathbf{x})$

procedure shown in Figure 5.18 (see Section 5.4) and denoted by $|B|$. The number of bigrams $|B|$ is bounded by $|U|^2$, which is shown in the figure as a dotted line. The grammar encodes a limited number of reordering operators, restricting the actual number of bigrams to a fraction of the theoretical bound.

Figure 5.23 shows some properties of the initial grammar $G(\mathbf{x})$ as a function of the input length. Both axes are shown in log scale so that we can more easily compare the curves in terms of their polynomial growth. The curve marked with triangles represents the number of unigrams in $G(\mathbf{x})$. The dashed curve marked with black circles represents the number of rules in $G(\mathbf{x})$. The dotted curve marked with red circles (the top-most curve) is the upperbound on the number of bigrams in $G(\mathbf{x})$, that is, the squared of $|U|$. Finally, the starred line represents $|B|$. In this figure we can see more clearly how the number of rules grows with the input length faster than the number of unigrams or bigrams do.

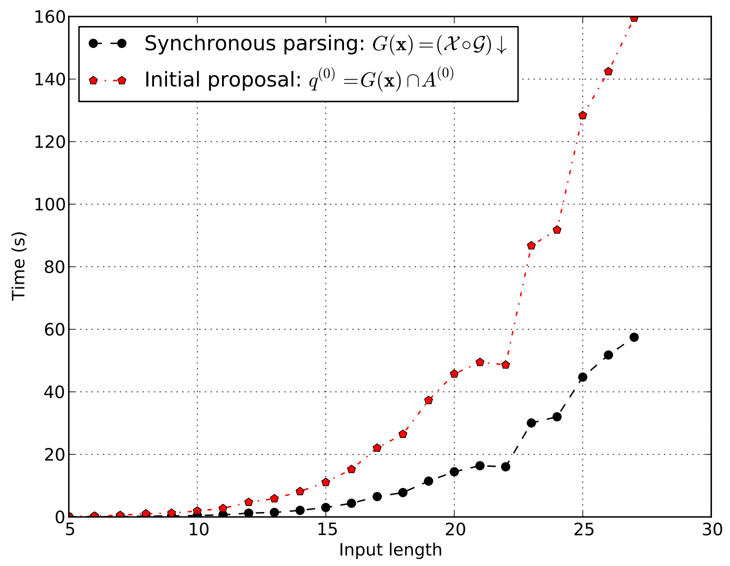


Figure 5.24: Time spent on building the initial proposal

Finally, before we focus on the refinement operations in different modes of OS*, Figure 5.24 shows the average time necessary to build the initial proposal. The dashed curve represents the time necessary to construct the translation hypergraph, that is, to obtain $G(\mathbf{x}) = (\mathcal{X} \circ \mathcal{G}) \downarrow$. The dotted curve shows the time to perform the initial intersection which incorporates the unigram upperbound weights, that is, $q^{(0)} = G(\mathbf{x}) \cap A^{(0)}$. Recall that the automaton $A^{(0)}$ is the simplest of our refinements. It contains a single state which is both initial and final and reweights the grammar without changing its size. Therefore, the dotted curve in Figure 5.24 can be thought of as a lowerbound on the time to be spent on each and every refinement after $A^{(0)}$. Recall that the intersection procedure runs in time proportional to $|R|$, the number of rules in the grammar, which is cubic with the length of the input (see Figure 5.19).

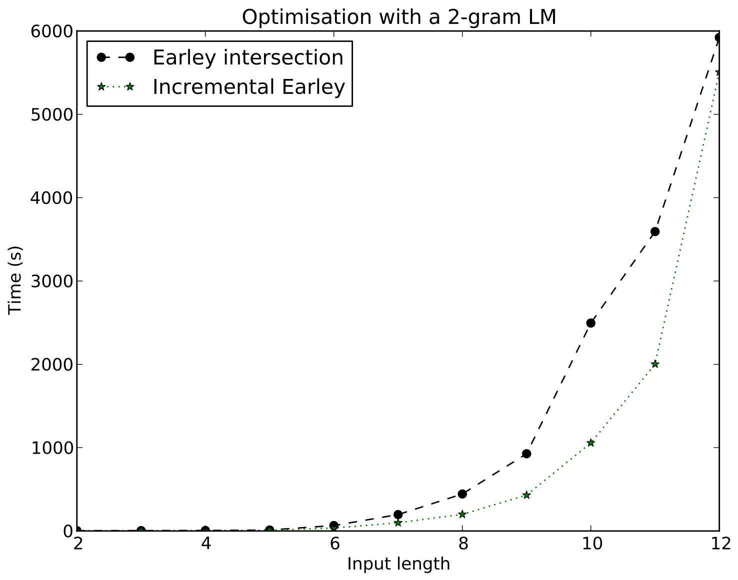


Figure 5.25: Incremental Earley intersection in optimisation

5.5.1 Optimisation

We start by discussing some results with a bigram language model and then experiment with language model distributions of increasing order. Let us first compare our incremental Earley intersection (see Section 4.2.3) to the original procedure by Dyer (2010). Figure 5.25 shows the average time necessary for convergence to the true optimum in optimisation with a bigram language model. The dashed curve marked with circles represents OS* running with the standard Earley intersection algorithm. The dotted curve marked with stars represents OS* running with our incremental variant. We observe a consistent gain in performance. However, this gain does not seem to change the order of the complexity of the algorithm, it rather scales it down by a constant.

In Figures 5.26, 5.27 and 5.28 we show the average number of refinements (or iterations), the time necessary for convergence to the true optimum, and the growth of the grammar (measured as a ratio between the size of the final

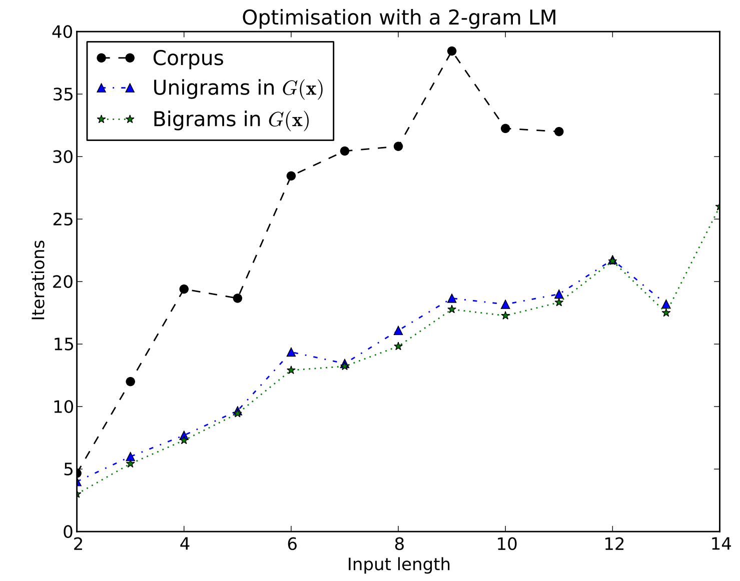


Figure 5.26: Number of refinements in optimisation with a 2-gram LM

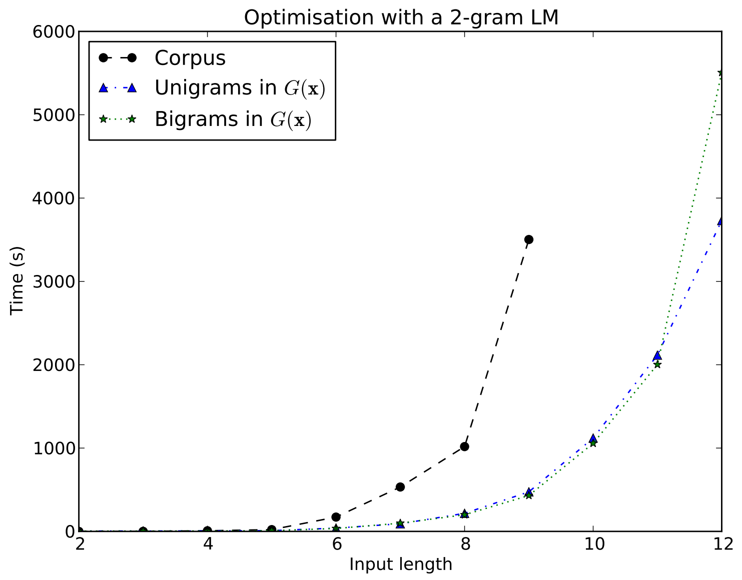


Figure 5.27: Time before convergence in optimisation with a 2-gram LM

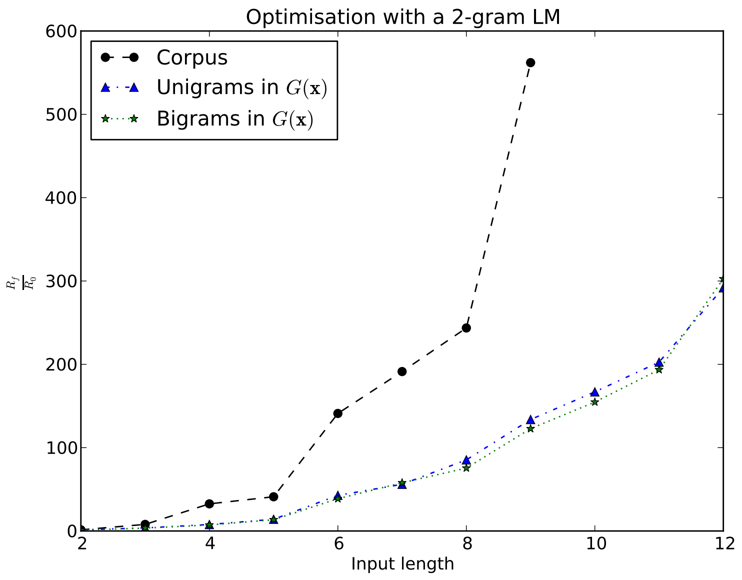


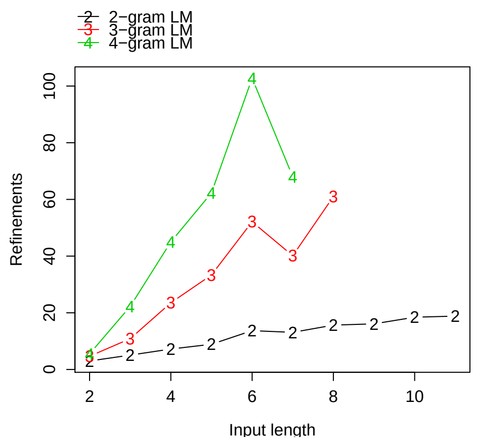
Figure 5.28: Grammar growth in optimisation with a 2-gram LM

rule set and the initial one), respectively. All figures represent optimisation with a bigram LM distribution. Moreover, the incremental variant of the intersection procedure was used. We compare three different scenarios. The dashed curves marked with circles (**Corpus**) represent OS* running with a corpus-level upperbound (see Section 4.2.2). The other curves represent OS* running with tighter upperbounds produced on a sentence basis. Recall from Section 4.2.2.2 that the upperbound distribution can be tightened with respect to unigrams (curves marked with triangles) and with respect to bigrams in $G(\mathbf{x})$ (curves marked with stars). As discussed before (see Section 4.3), there is a large gain in tightening the upperbounds to the active vocabulary of unigrams. A much smaller gain can be achieved using the active vocabulary of bigrams. Figure 5.28 stresses the fact that tightening the LM upperbound distribution leads to OS* exploring a smaller space before convergence. Note how the final grammars are much smaller with tighter upperbounds.

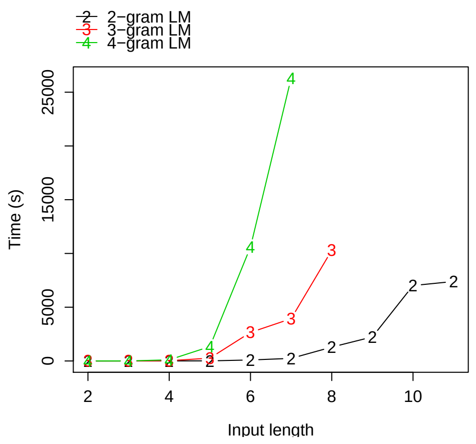
Figure 5.26 shows that we incorporate just a fraction of all bigrams in the grammar. Consider the sentences with 10 words. Their grammars contain on average 150 unigrams (see Figure 5.21) and 5000 bigrams (see Figure 5.22). In these cases, to obtain an optimum derivation with a certificate of optimality, OS* incorporates on average less than 20 unigram contexts (out of the 150 available). We will see that for higher-order LMs, the fraction of the complete space of n-grams explicitly incorporated by refinements is even smaller. While the number of refinement operations is encouragingly small, the decoding time is an issue. Each refinement requires a general intersection procedure which is at least proportional to the size of the current grammar $|R|$. Although this factor is alleviated by our incremental intersection, the decoding time still grows quickly.

We now turn the discussion to optimisation with language model distributions of increasing order. Figure 5.29 shows the average number of refinements and the average time necessary for convergence to the true optimum. The current implementation faces timeouts depending on the length of the input sentence and the order of the language model, explaining why certain curves are interrupted earlier than others in Figure 5.29. At this point we are not relying on selective intersection, therefore, each time a k-gram context is incorporated, it is incorporated everywhere in the hypergraph.

The number of refinements up to convergence appears to be linear with the input length (see Figure 5.29a), while the total duration grows much faster (see Figure 5.29b). Remember that at each iteration we perform the intersection between the current proposal (a wCFG) and a refinement (a small recursive wFSA). This intersection runs in time proportional to $|R||Q|^r$, where $|R|$ is the size of the rule set of the grammar (initially proportional to I^3), r is the length of the longest rule plus one, and $|Q|$ is the number of states in the refinement (typically small and proportional to the order of the refinement). OS* seems very economic in that only n-grams that are proven necessary are explicitly intersected with the grammar. Observe how



(a) Number of refinements



(b) Time for convergence.

Figure 5.29: OS* in optimisation mode: number of refinements and time for convergence to true optimum. Each curve represents optimisation with LM distributions of different order: (2) uses an upperbound on a 2-gram LM; (3) uses an upperbound on a 3-gram LM; and (4) uses an upperbound on a 4-gram LM.

we need very few iterations to converge to the optimum. However, successive refinement operations turn out to become costly as the algorithm progresses and the grammar grows. One of the reasons is associated with the fact that the grammar is refined everywhere, an observation that motivates the idea of selective intersection. Another reason relates to the fact that regardless of how local the refinements are, the general intersection procedure is still proportional to $|R|$, which in turn is proportional to I^3 (see Figures 5.19 and 5.23). This observation motivates the idea of performing incremental intersections.

Table 5.2 shows some important quantities regarding optimisation with OS* using a 4-gram LM. The first column shows how many sentences we are considering, the second column shows the sentence length, the third column m is the average number of refinements up to convergence. Column $|A^{(t)}|$ refers to the refinement type, which is the number of states in the automaton

S	Length	m	$\|A^{(t)}\|$	count	$\frac{\|R_f\|}{\|R_0\|}$
9	4	45.0	2	20.3	74.6 ± 53.9
			3	19.2	
			4	5.4	
10	5	62.3	2	21.9	145.4 ± 162.6
			3	32.9	
			4	7.5	
9	6	102.8	2	34.7	535.8 ± 480.0
			3	54.9	
			4	13.2	

Table 5.2: Optimisation with a 4-gram LM.

$A^{(t)}$, that is, the order of the n-grams being re-weighted (e.g. $|A^{(t)}| = 2$ when refining bigrams sharing a one-word context). Column *count* refers to the average number of refinements that are due to each refinement type. Finally, the last column compares the number of rules in the final proposal to that of the initial one.

Let us start by focusing on how much context OS* needs to take into account for finding the optimum derivation. Table 5.2 (column m) shows that OS* explores a very reduced space of n-gram contexts up to convergence. To illustrate that, consider the last row in Table 5.2 (sentences with 6 words). On average, convergence requires incorporating only about 103 contexts of variable order, of which 55 are bigram (2-word) contexts (remember that $|A^{(t)}| = 3$ when accounting for a 2-word context). On average, in sentences with 6 words, about 2,000 bigrams are compatible with strings generated by $G(\mathbf{x})$. This means that only 2.75% of these bigrams (55 out of 2,000) need to be explicitly accounted for. This illustrates how wasteful a full intersection would be.

A problem, however, is that the time until convergence grows quickly with the length of the input (Figure 5.29b). This can be explained as follows. At each iteration the grammar is refined to account for n-grams sharing a context of $(n-1)$ words. That operation typically results in a larger grammar: most

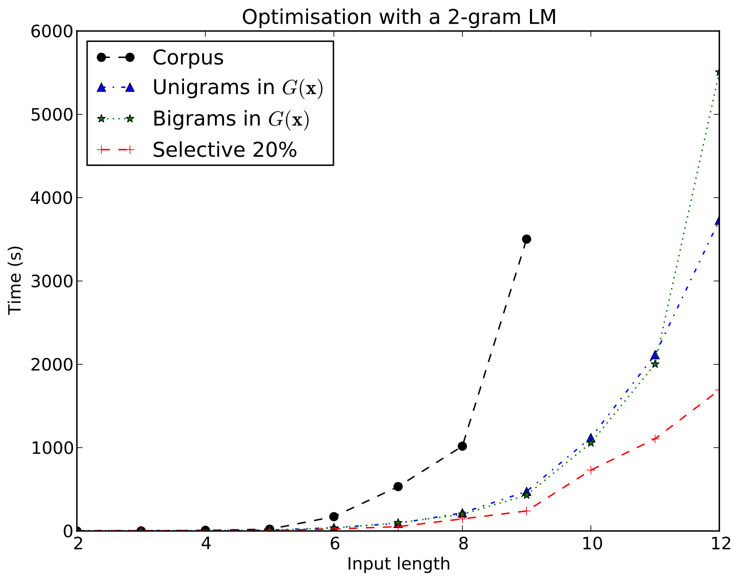


Figure 5.30: Impact of selective intersection on time for convergence in optimisation with a 2-gram LM

rules are preserved, some rules are deleted, but more importantly, some rules are added to account for the portion of the current grammar that involves the selected n-grams. Enlarging the grammar at each iteration means that successive refinements become incrementally slower.

The histogram of refinement types, shown in Table 5.2, highlights how efficient OS* is with respect to the space of n-grams it needs to explore before convergence. The problem is clearly not the number of refinements, but rather the relation between the growth of the grammar and the successive intersections.

Before we turn to sampling, let us discuss some results concerning selective intersection. At this point we experiment with a 2-gram language model. The dashed curves marked with crosses in Figures 5.30 and 5.31 show how decoding time and grammar growth are reduced by running OS* with selective intersection. In this experiment, the algorithm anticipates computation for the instances that hold $h = 20\%$ of the probability mass associated with

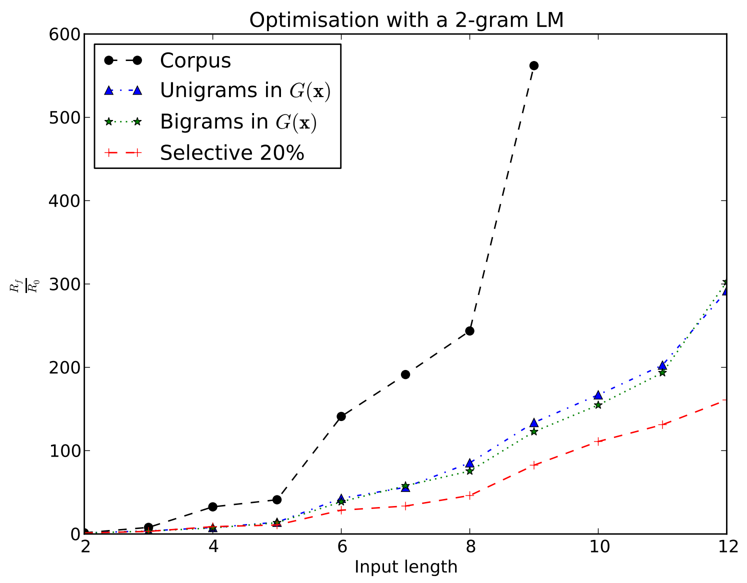


Figure 5.31: Impact of selective intersection on the growth of the grammar in optimisation with a 2-gram LM

a certain refinement type. Marginals are estimated using the Inside-Outside algorithm and recomputed whenever an instance of a previously incorporated type redeems itself from the group of low probability instances (see Section 5.3.2). We observe a consistent improvement in decoding time and grammar growth. Performance varies with the parameter h. Larger h reduces the number of iterations before convergence at the cost of growing the grammar quickly. A procedure for the tuning of this parameter is reserved for future work.

5.5.2 Sampling

Let us turn to results in sampling mode with language model distributions of increasing order. Figure 5.32 shows the average number of refinements and the average time necessary for convergence to different levels of acceptance rate. Figure 5.32a shows that sampling is more economic than optimisation

S	Input	m	$\lvert A^{(t)}\rvert$	count	$\frac{\lvert R_f\rvert}{\lvert R_0\rvert}$
10	5	1.0	2	1.0	1.9 ± 1.0
10	6	6.6	2	6.3	17.6 ± 13.6
			3	0.3	
10	7	14.5	2	12.9	93.8 ± 68.9
			3	1.5	
			4	0.1	

Table 5.3: Sampling with a 4-gram LM and reaching a 5% acceptance rate.

in that it explicitly incorporates even fewer contexts. Note how OS* converges to acceptance rates from 1% to 10% in much fewer iterations than the necessary to find an optimum.[14] Although the convergence in sampling takes fewer iterations than in optimisation, the total time necessary to converge to the desired acceptance rate is still an issue (Figure 5.32b).

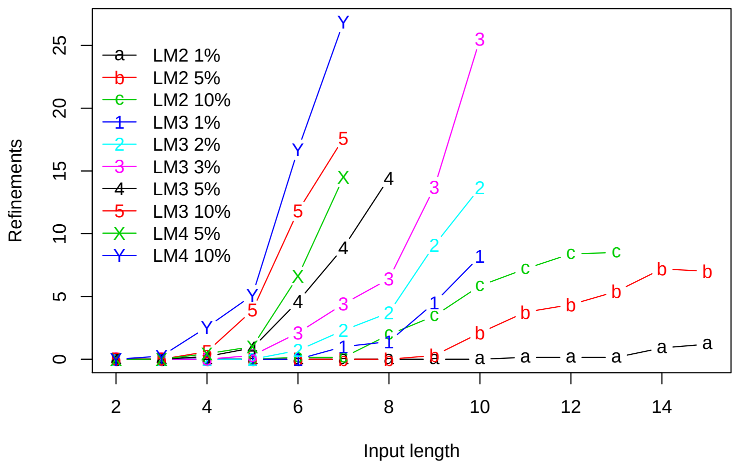
Table 5.3 shows the same quantities as Table 5.2, but now for sampling. It is worth highlighting that even though we are using an upperbound over a 4-gram LM (and aiming at a 5% acceptance rate), very few contexts are selected for refinement, most of them lower-order ones (see rows with $|A^{(t)}| = 2$). This suggests that it is possible to design a sampler for a distribution of order n without actually incorporating n-grams. Once we improve the performance of the refinement operations, a direction for future work is to investigate the use of a sampler designed for a distribution p_1 to sample in a rejection sampling framework from a more complex distribution p_2. One application of this idea is to design a sampler for a 2-gram LM distribution and sample from a model which incorporates a 4-gram LM component. Another application is to add to the parameterisation some global feature which requires a complete derivation to be assessed. We can design a sampler for a simpler

[14]Currently we use MERT to train the model's weight vector, which is normalised by its L_1 norm in the Moses implementation. While optimisation is not sensitive to the scale of the weights, in sampling the scale determines how flat or peaked the distribution is. (Arun et al., 2010) experiment with scaling MERT-trained weights as to maximise BLEU on held-out data, as well as with MBR training. A more adequate training algorithm along similar lines is reserved for future work.

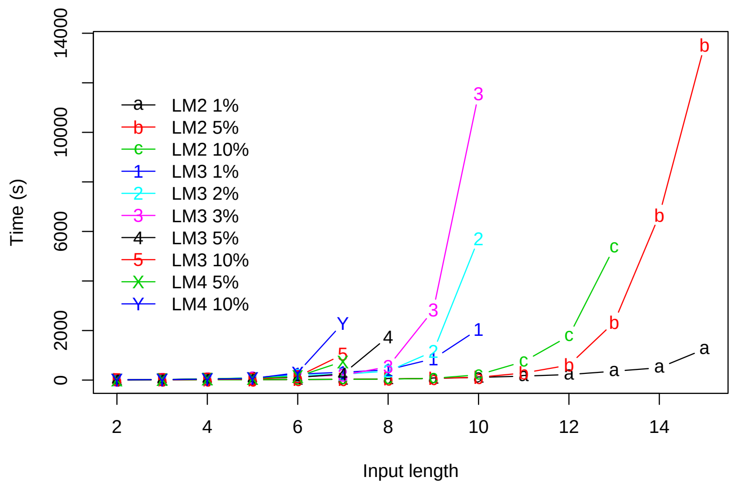
parameterisation which overlooks such a feature achieving a certain acceptance rate with OS*. We then use the resulting sampler to perform rejection sampling in relation to the more complex distribution at the cost of some decrease in the acceptance rate.

Observe that an improved acceptance rate always leads to faster acquisition of exact samples after we stop refining our proxy distribution. However, Figure 5.32b shows for example that moving from 5% to 10% acceptance rate using a 4-gram LM (curves X and Y) is time-consuming. Thus, there is a trade-off between how much time we spend improving the acceptance rate and how many exact samples we intend do draw. Figure 5.33 shows the average time to draw batches between one and one million samples from exact samplers that were refined up to 5% and 10% acceptance rate, respectively. The samplers at 5% AR (which are faster to obtain) turn out to be more efficient if we aim at producing less than 10 thousand samples.

Finally, note that samples are independently drawn from the final proposal, making the approach an appealing candidate to parallelism in order to increase the effective acceptance rate. That is, if we independently sample from N copies of an exact sampler with acceptance rate a we effectively multiply the acceptance rate by N.



(a) Number of refinements



(b) Time for convergence

Figure 5.32: OS* in sampling mode: number of refinements and time for convergence to different acceptance rates: (a, b and c) use a 2-gram LM to reach 1, 5 and 10% AR; (1-4) use a 3-gram LM to reach 2, 3, 5 and 10% AR; and (X, Y) use a 4-gram LM to reach 5 and 10% AR.

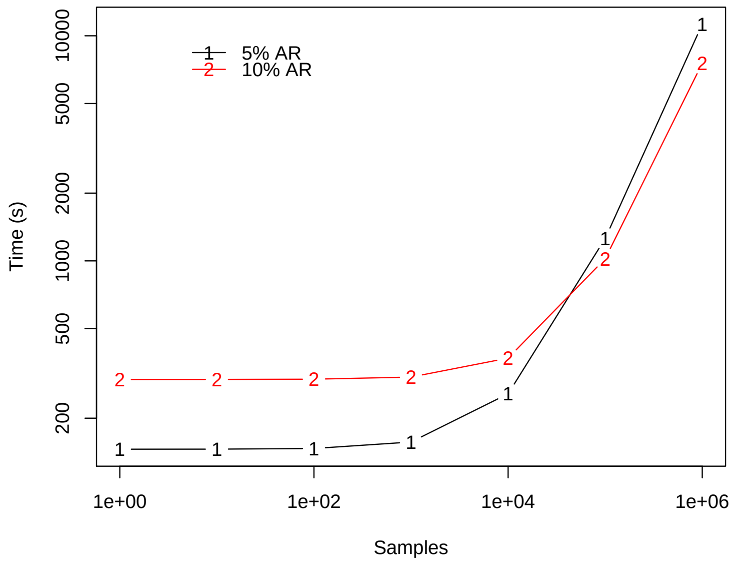


Figure 5.33: Average time to draw 1 to 1 million samples, for input sentences of length 6, using a 4-gram LM at 5% (curve 1) and 10% (curve 2) acceptance rate (including the time to produce the sampler).

Chapter 6

Phrase-based translation with OS*

In this chapter we introduce an OS* approach to exact inference for phrase-based translation models such that of Koehn et al. (2003). In hierarchical SMT (see Chapter 5) the core of the complexity issue is the intersection between the translation forest (a wCFG) and the language model (a wFSA). In phrase-based translation, besides the expensive intersection between the translation lattice (a wFSA) and the language model, reordering is not hard-constrained by the model of translational equivalences. Thus, in the general case, a translation can align to any of the input's many permutations.[1] A key constraint in phrase-based decoding is that each input word must be translated exactly once (we call it the *non-overlapping constraint*).[2] Encoding this constraint requires an exponential number of states, therefore, unless some reordering limit is enforced, the translation lattice $G(\mathbf{x})$ is intractable even before the intersection with the language model.

Following the OS* approach to SMT introduced in Chapter 4, we design a tractable upperbound to the goal distribution and refine this initial proposal based on evidence gathered by rejection sampling. In order to make $G(\mathbf{x})$ tractable, we relax the space of translation derivations with respect to the non-overlapping constraint. In a nutshell, the proposal lattice does not incorporate the exponentially many constraints that prevent overlaps. Instead it incorporates a looser constraint that states that a derivation must cover

[1]A sentence of length I has $I! = \prod_{i=1}^{I} i$ permutations.

[2]However, Roth et al. (2010) propose a model to work with overlapping phrases and inference is based on Gibbs sampling.

as many positions as there are words in the input. However, overlaps might happen and some words might be translated multiple times while others are never translated. We proceed by rejection sampling and refine the proposal on the basis of two different types of evidence: (1) if a rejected derivation is valid, in that each input word is translated exactly once, we refine the proposal by incorporation of some k-gram context effectively accounting for part of the intersection with a full language model; (2) if a rejected derivation is invalid, in that at least one input word is translated multiple times, we incorporate to the proposal some constraints that prevent certain overlaps to happen.[3]

Another difference with respect to hierarchical models concerns the parameterisation of the model. Recall from Section 2.4.2 that phrase-based models use a notion of *distortion cost* δ, which depends on the interactions between two decisions, namely, the relative position of the current biphrase with respect to the last source position covered by the most recently chosen biphrase. In order to directly incorporate distortion costs to the initial proposal, we start from a translation lattice whose states encode sufficient information to compute δ upfront.

Novel contributions of this chapter are:

- a new approach for **exact inference** (optimisation and sampling) for phrase-based translation models (Section 5.2);
- novel tractable **upperbounds** on the translation lattice.

6.1 Phrase-based SMT

In phrase-based translation the underlying model is represented by a wFST, that is, $\mathcal{G}$ in $G(\mathbf{x}) = (\mathcal{X} \circ \mathcal{G}) \downarrow$ (see Section 2.5) is a wFST. The translation

[3]This type of relaxation is shared with methods based on ILP and Lagrangian relaxation (see Section 3.3.2).

hypergraph $G(\mathbf{x})$ (also known as a translation lattice) is obtained by applying $\mathcal{G}$ to the input sentence, in other words, by the composition between the input $\mathbf{x} = \langle x_1, \ldots, x_I \rangle$, represented as the identity-transducer $\mathcal{X}$, and a (cascade of) finite-state transducer(s) $\mathcal{G}$ (Kumar and Byrne, 2003; Kumar et al., 2006). Because $\mathcal{X}$ is acyclic and our phrase-based model does not allow unbounded insertion (Koehn et al., 2003), the resulting lattice $G(\mathbf{x})$ can be represented by an acyclic wFSA.

In this section, weighted deduction (or weighted logic) is used to compactly formalise spaces of translation derivations.[4] A logic program, such as the one shown in Figure 6.1, implicitly defines a weighted lattice such that: (1) each state represents the signature of an item of the program; (2) each deduction defines a transition, whose origin is the state associated with the signature of the antecedent and the destination is the state associated with the signature of the consequent, and whose weight is given by the weight of the consequent; (3) the initial state is an empty hypothesis associated with a null antecedent (used in axioms); and (4) final states are those whose signatures are associated with goal items.[5]

Figure 6.1 is the weighted logic that implements $G(\mathbf{x})$ for an input sentence of length I. In the program, $G(\mathbf{x})$ is parameterised without a language model, that is, a transition is weighted by the translation features ϕ, which are local to the biphrase, and by the distortion cost δ. An item in the program has the form $[i, V, \gamma]$, where i is the last source position covered by the most recently chosen biphrase, V is a coverage bit vector of length I, and γ is a target phrase. An instantiated biphrase $\langle x_i^{i'}, \gamma \rangle$, also denoted by a rule item $R(x_i^{i'}, \gamma)$, is such that it covers the input span $i \ldots i'$ producing a target phrase γ. A rule item $R(x_i^{i'}, \gamma)$ is a short for $\langle x_i^{i'}, \gamma \rangle \in R$, where the collection

[4] Weighted deduction is discussed in Section 5.3. For a more comprehensive discussion refer to (Goodman, 1998). Lopez (2009) formalises several phrase-based search spaces using weighted deduction.

[5] Recall that a hypergraph with multiple goal nodes can be easily converted into a hypergraph with a single goal node (see Section 2.3).

Goal $\quad [i \in \{1 \ldots I\}, 1^I, \gamma]$

Axioms

$$\frac{R(x_{i+1}^{i'}, \gamma)}{[i', 0^i 1^{i'-i} 0^{I-i'}, \gamma] : \delta(0, i+1) + \phi(\langle x_{i+1}^{i'}, \gamma \rangle)}$$

Concatenate

$$\frac{[i'', V, \gamma'] \; R(x_{i+1}^{i'}, \gamma)}{[i', V \vee 0^i 1^{i'-i} 0^{I-i'}, \gamma] : \delta(i'', i+1) + \phi(\langle x_{i+1}^{i'}, \gamma \rangle)} \quad V \wedge 0^i 1^{i'-i} 0^{I-i'} = 0^I$$

Figure 6.1: Phrase-based translation lattice (without LM)

R is the implicit set of instantiated biphrases.[6]

A deduction of the kind $\frac{[i,V,\gamma]}{[i',V',\gamma']:\omega}$ is associated with a transition $e : \langle i, V, \gamma \rangle \xrightarrow{\gamma':\omega} \langle i', V', \gamma' \rangle$ where $t[e] = \langle i, V, \gamma \rangle$ is the origin state (or tail), $h[e] = \langle i', V', \gamma' \rangle$ is the destination state (or head), $w[e] = \omega$ is the edge's weight and $i[e] = \gamma'$ is the edge's label.[7] The Goal of the program is to prove items that cover all source positions exactly once. The non-overlapping constraint is represented by the side condition $V \wedge 0^i 1^{i'-i} 0^{I-i'}$.[8] The operator $\wedge$ is a bitwise AND and $0^i 1^{i'-i} 0^{I-i'}$ represents a bit vector of length I, where the first i and the last $I - i'$ bits are set to 0, and the inner $i' - i$ bits are set to 1 (representing the source phrase being covered). This bit vector represents an update rule — note how the consequent in Concatenate updates V using the bitwise OR $\vee$. This program constructs the space of all derivations that translate each input word exactly once weighted only locally (however including the distortion cost).

There is a finite number of target phrases per input phrase, and this number is typically limited by a constant B. The space complexity of the

[6]This set can be seen as the implicit intersection between the input (encoded as an identity-transducer) and an unweighted FST representing the phrase table.

[7]With some abuse of language we use $\langle i, V, \gamma \rangle$ to denote the state in the lattice associated with the item whose signature is $[i, V, \gamma]$.

[8]In this representation, let v^m represent m repetitions of v if $m > 0$, otherwise, v^m returns a vector of length 0.

program in Figure 6.1 is $O(I^3 2^I B)$, by inspection of the free variables in the program. A linear factor I is due to the need to track down the last source position that has been covered. A quadratic factor $I^2 B$ is due to the number of segmentations of the input and their translations, however, with a maximum phrase length L this factor is lowered to IL^2B. Finally, the bit vector leads to an exponential factor 2^I, which makes $G(\mathbf{x})$ intractable in the general case. Lopez (2009) documents a number of strategies to limit reordering lowering the exponential dependency on input length to 2^d where d is a constant called *the distortion limit*.[9] A distortion limit characterises a form of pruning that acts directly in the generative capacity of the model. In this thesis we are interested in the original NP-complete formulation, i.e., **without a distortion limit**.

Decoding requires re-weighting the translations in $G(\mathbf{x})$ by the language model A. The space of solutions is given by the intersection $G_{\cap A} = G(\mathbf{x}) \cap A$, which is itself a wFSA and represents the goal distribution p over which we want to perform inference. The intersection with an n-gram language model adds a multiplicative factor I^{n-1} to the complexity we have just discussed.[10] This intersection is prohibitive to compute exactly, except for short sentences using a low-order language model. Recall that the number of states of A grows exponentially with the order n of the language model. Moreover, unless a distortion limit rules out certain permutations of the input, the translation lattice $G(\mathbf{x})$ grows exponentially with the length of the input f.

[9]One strategy is to impose a maximum distortion d. That is, a biphrase that covers $x_i^{i'}$ and extends the item $[i'', V, \gamma]$ must comply with $|i - i''| \leq d$. Another strategy, in use in the Moses toolkit (Koehn et al., 2007), establishes that the last source word covered by any biphrase must be within d words from the leftmost uncovered source position, i.e., $|i' - l| \leq d$, where l tracks this leftmost word. The interaction of the first strategy with histogram and threshold pruning leads to a high number of *dead hypotheses* in the beam-search dynamic program. That is, hypotheses that cannot be extended by any biphrase respecting the distortion limit. These dead hypotheses compromise decoding in that stacks are filled with hypotheses that can never lead to complete derivations (Lopez, 2009).

[10]The size of the target vocabulary depends linearly on the input length due to the limit on the maximum number of translations per source segment.

$$\textsc{Goal} \qquad [i \in \{1 \dots I\}, 1^I, \textsc{eos}\bullet, y_{J-n+3}^{J}\,\textsc{eos}]$$

$$\textsc{Axioms}$$
$$\frac{R(x_{i+1}^{i'}, y_1\gamma)}{[i', 0^i 1^{i'-i} 0^{I-i'}, y_1 \bullet \gamma, y_{3-n}^{1}] : w_A}$$
$$\text{where} \quad y_{j\leq 0} = \textsc{bos}$$
$$w_A = \phi(\langle x_{i+1}^{i'}, y_1\gamma\rangle) + \delta(0, i+1) + \psi(y_1 | y_{1-n}^{0})$$

$$\textsc{Concatenate}$$
$$\frac{[i'', V, \gamma'\bullet, y_{j-n+1}^{j-1}] \; R(x_{i+1}^{i'}, y_j\gamma)}{[i', V \vee 0^i 1^{i'-i} 0^{I-i'}, y_j \bullet \gamma, y_{j-n+2}^{j}] : w_C} \quad V \wedge 0^i 1^{i'-i} 0^{I-i'} = 0^I$$
$$\text{where} \quad w_C = \phi(\langle x_{i+1}^{i'}, y_j\gamma\rangle) + \delta(i'', i+1) + \psi(y_j | y_{j-n+1}^{j-1})$$

$$\textsc{Scan}$$
$$\frac{[i', V, \gamma' \bullet y_j\gamma, y_{j-n+1}^{j-1}]}{[i', V, \gamma' y_j \bullet \gamma, y_{j-n+2}^{j}] : \psi(y_j | y_{j-n+1}^{j-1})}$$

$$\textsc{Accept}$$
$$\frac{[i, V, \gamma\bullet, y_{J-n+2}^{J}]}{[i, V, \textsc{eos}\bullet, y_{J-n+3}^{J}\,\textsc{eos}] : \psi(\textsc{eos} | y_{J-n+2}^{J})}$$

Figure 6.2: Full search space in phrase-based SMT

The weighted logic in Figure 6.2, adapted from (Lopez, 2009), implements $G_{\cap A}$ exactly.[11] An item in the program is denoted by $[i, V, \gamma' y_j \bullet \gamma, y_{j-n+1}^{j-1}]$, where (1) i tracks the last position covered in the input; (2) V is a coverage bit vector of length I; (3) $\gamma' y_j \bullet \gamma$ is a target phrase whose prefix $\gamma' y_j$ has been scanned and γ is yet to be scanned; and (4) y_{j-n+1}^{j} is a context history representing the $n-1$ words preceding the dot (it might contain a string longer than $\gamma' y_j$). Scanning concerns the intersection with an n-gram language model, that is, scanned words have their LM weights incorporated with respect to the preceding $n-1$ target words (observe the component ψ

[11]Recall that a wFSA is an instance of a hypergraph, therefore the weighted logic in Figure 5.1 (Earley intersection) also implements $G_{\cap A}$ exactly. The program in Figure 6.2 is discussed to highlight the connections with the dynamic program of a beam-search procedure, where an n-gram target language model is instantiated only implicitly.

contributing to the weight of the items in the program). In this program there are two types of transitions: (1) those that select a biphrase consuming an input span and producing the first word of the target phrase, and (2) those that scan through the remaining target words within the phrase pair without consuming additional input words. The former corresponds to a deduction of the kind

$$\frac{[i, V, \gamma \bullet, y_{j-n+1}^{j-1}]}{[i', V', y_j \bullet \gamma', y_{j-n+2}^{j}] : \omega} ,$$

which is associated with a transition e, such that $t[e] = \langle i, V, \gamma \bullet, y_{j-n+1}^{j-1} \rangle$, $h[e] = \langle i', V', y_j \bullet \gamma', y_{j-n+2}^{j} \rangle$, $w[e] = \omega$ and $i[e] = y_j$. The latter corresponds to a deduction of the kind

$$\frac{[i, V, \gamma' \bullet y_j \gamma, y_{j-n+1}^{j-1}]}{[i, V, \gamma' y_j \bullet \gamma, y_{j-n+2}^{j}] : \omega} ,$$

which is associated with a transition e, such that $t[e] = \langle i, V, \gamma' \bullet y_j \gamma, y_{j-n+1}^{j-1} \rangle$, $h[e] = \langle i, V, \gamma' y_j \bullet \gamma, y_{j-n+2}^{j} \rangle$, $w[e] = \omega$ and $i[e] = y_j$.

In analogy to standard dynamic programming, the AXIOMS extend the null hypothesis by selection of a biphrase. Partial hypotheses whose target phrases have been completely scanned, represented by items where the dot is at the end of the phrase, are extended from left to right in target language order by a non-overlapping biphrase (see CONCATENATE). The operation SCAN moves the dot forward in the phrase incorporating LM weights and updating the context history. The operation ACCEPT incorporates the weight of the end-of-sentence symbol (EOS) in context. The GOAL is to translate every input word exactly once, incorporating the LM weights of all target n-grams. This program builds the fully parameterised space of all derivations that translate each input word exactly once. In the next sections we discuss how we relax $G(\mathbf{x})$ making it tractable and how a proxy to $G_{\cap A}$ is built and incrementally refined following the methodology introduced in Chapter 4.

6.2 An OS* approach to phrase-based SMT

Recall that the full intersection $G(\mathbf{x}) \cap A$ defines a weighted set of translation derivations $\langle \mathcal{D}, p \rangle$, where p is an unnormalised goal distribution p. Derivations in $\mathcal{D}$ comply with the non-overlapping constraint imposed by our model of translational equivalence and encoded in $G(\mathbf{x})$. We introduce an approach where instead of explicitly constructing the full intersection $G(\mathbf{x}) \cap A$ we incrementally produce a sequence of "proposal" lattices $q^{(t)}$, all of which upperbound p, where $q^{(0)} = G'(\mathbf{x}) \cap A^{(0)}$, $q^{(1)} = q^{(0)} \cap A^{(1)}$, $\ldots$, $q^{(t+1)} = q^{(t)} \cap A^{(t+1)}$, etc.[12] $G'(\mathbf{x})$ is a relaxation of $G(\mathbf{x})$ which does not enforce the non-overlapping constraint everywhere in the lattice. That is, $G'(\mathbf{x})$ includes at least all derivations in $G(\mathbf{x})$, but it also includes invalid derivations containing overlapping phrases. $A^{(0)}$ is an upperbound on the language model distribution, that is, it is an optimistic, low complexity version of the automaton A which forgets the contexts of the n-grams. Each proposal distribution defines a weighted set of translation derivations $\langle \mathcal{D}', q^{(t)} \rangle$. This set is such that $\mathcal{D} \subseteq \mathcal{D}'$ and $q^{(t)}$ upperbounds p everywhere, i.e., $q^{(t)}(\cdot) \geq p(\cdot)$. Each increment $A^{(t)}$ is a small automaton that either: (1) incorporates to $q^{(t-1)}$ some specific k-gram context (i.e. sequence of k words) not yet made explicit in the previous increments, where k takes some value between 1 and n; or (2) constrains $q^{(t-1)}$ preventing some overlaps from happening, which has the effect of removing from $q^{(t-1)}$ some of its invalid derivations. This process produces a sequence of proposals such that $q^{(0)}(\cdot) \geq q^{(1)}(\cdot) \geq q^{(2)}(\cdot) \geq \cdots \geq p(\cdot)$.

In Section 6.2.1, we introduce different strategies to produce a simple and compact lattice $G'(\mathbf{x})$ in the absence of a distortion limit. The weighted logic shown in Figure 6.3 implements one such strategy. An item in the program is denoted by $[\langle x_i^{i'}, \gamma' y_j \bullet \gamma \rangle, c]$, where: (1) $\langle x_i^{i'}, \gamma' y_j \bullet \gamma \rangle$ is a dotted

[12]With some abuse of language we refer to the distribution meaning the weighted set and write $q^{(t)} \cap A^{(t+1)}$.

Goal $\quad [\langle x_i^{i'}, \gamma \bullet \rangle, I]$

Axioms

$$\frac{R(x_i^{i'}, y_1\gamma)}{[\langle x_i^{i'}, y_1 \bullet \gamma\rangle, i' - i + 1] : \phi(\langle x_i^{i'}, y_1\gamma\rangle) + \delta(0, i)}$$

Concatenate

$$\frac{[\langle x_l^{l'}, \gamma' \bullet \rangle, c] \; R(x_i^{i'}, y_j\gamma)}{[\langle x_i^{i'}, y_j \bullet \gamma\rangle, c + i' - i + 1] : \phi(\langle x_i^{i'}, y_j\gamma\rangle) + \delta(l', i)} \quad \begin{array}{l} i > l' \vee i' < l \\ c + i' - i + 1 \leq I \end{array}$$

Scan

$$\frac{[\langle x_i^{i'}, \gamma' \bullet y_j\gamma\rangle, c]}{[\langle x_i^{i'}, \gamma' y_j \bullet \gamma\rangle, c] : \bar{1}}$$

Figure 6.3: $G(\mathbf{x})$: relaxed translation lattice

biphrase whose prefix $\gamma' y_j$ has already been scanned and the suffix γ is yet to be scanned; and (2) c is the number of source words covered thus far. The Goal of the program is to prove items that cover I input words. The Axioms extend the null hypothesis by addition of one phrase pair. Note that the language model is not part of the parameterisation at this point. In Concatenate, hypotheses are extended from left to right in target language order by selection of phrase pairs that do not overlap with the immediately preceding choice of biphrase. If the spans $x_l^{l'}$ and $x_i^{i'}$ are covered by the previous and the current biphrase, respectively, then the side condition $i > l' \vee i' < l$ prevents adjacent transitions from overlapping. The side condition $c + i' - i + 1 \leq I$ prevents the construction of derivations covering more than I words. Observe that there is no coverage bit vector preventing overlaps beyond the context of adjacent transitions. The operation Scan moves the dot forward within a biphrase. This program constructs the space of all derivations that translate exactly I words weighted only locally (however including the distortion cost).

By inspection of the free variables in the program, the search space complexity is $O(I^6B)$, where: (1) a linear term I is due to the need to track the number of words translated; (2) another linear term IB concerns the target phrases; (3) a quadratic term I^2 is due to the need to track the previous biphrase; and (4) another quadratic term I^2 is due to the current biphrase. With a maximum phrase length L, (3) and (4) are both lowered to linear terms IL^2. At the cost of producing derivations that would not be defined in $G(\mathbf{x})$, this relaxed space has no exponential dependency on I (see Section 6.2.1 for other definitions of $G'(\mathbf{x})$).

Consider an enriched terminal symbol represented as a triple $\langle b, d, c\rangle$, where: (1) b is an instantiated biphrase such as $\langle x_i^{i'}, y_{j+1}^{j+k}\rangle$; (2) d refers to a dot positioned adjacently to the right of y_{j+d} in y_{j+1}^{j+k}; and (3) c is the number of source words covered thus far. Similarly to the program in Figure 6.2, the program in Figure 6.3 defines two types of transitions. The transitions that *begin a biphrase* correspond to deductions of the kind

$$\frac{[\langle x_l^{l'}, \gamma' \bullet \rangle, c']}{[\langle x_i^{i'}, y_{j+1} \bullet \gamma\rangle, c] : \omega} \,.$$

Each such deduction (see CONCATENATE) is associated with a transition e such that $t[e] = \langle\langle x_l^{l'}, \gamma' \bullet\rangle, c'\rangle$, $h[e] = \langle\langle x_i^{i'}, y_{j+1} \bullet \gamma\rangle, c\rangle$, $w[e] = \omega$ and $i[e] = \langle b, 1, c\rangle$, where $\langle b, 1\rangle$ represents the dotted biphrase $\langle x_i^{i'}, y_{j+1} \bullet \gamma\rangle$. The other transitions correspond to deductions of the kind

$$\frac{[\langle x_i^{i'}, \gamma' \bullet y_{j+d}\gamma\rangle, c]}{[\langle x_i^{i'}, \gamma' y_{j+d} \bullet \gamma\rangle, c] : \omega} \,.$$

Each such deduction (see SCAN) is associated with a transition e such that $t[e] = \langle\langle x_i^{i'}, \gamma' \bullet y_{j+d}\gamma\rangle, c\rangle$, $h[e] = \langle\langle x_i^{i'}, \gamma' y_{j+d} \bullet \gamma\rangle, c\rangle$, $w[e] = \omega$ and $i[e] = \langle b, d, c\rangle$, where $\langle b, d\rangle$ represents the dotted biphrase $\langle x_i^{i'}, \gamma' y_{j+d} \bullet \gamma\rangle$.[13] Labelling edges with these triples, rather than target words alone, will be

[13]A biphrase b whose target phrase contains k words, in a partial derivation that covers c input words, motivates a sequence of k transitions $\langle\langle b, 1, c\rangle, \ldots, \langle b, k, c\rangle\rangle$, each one producing one of the k target words in the biphrase.

useful later when we introduce strategies to incrementally incorporate non-overlapping constraints to the proposals.

Consider an edge e labelled with a triple $i[e] = \langle b, d, c \rangle$, where $b = \langle x_i^{i'}, y_{j+1}^{j+k} \rangle$, $d \in [1 \dots k]$ and $c \in [1 \dots I]$. The following operators help characterise the edge: (1) $\mathrm{b}[e]$ returns the instantiated biphrase b, where $\mathrm{b}_x[e]$ refers to the input phrase and $\mathrm{b}_y[e]$ refers to the output phrase; (2) $\mathrm{d}[e]$ returns the position d of the dot, which refers to the word in $\mathrm{b}_y[e]$ produced by e; (3) $\mathrm{y}[e]$ returns the target word produced by the edge, that is, the word y_{j+d} to the left of the dot; (4) $\mathrm{c}[e]$ returns the number c of input words already covered; (5) $\mathrm{x}_i[e]$ returns 1 if e covers input position i, i.e. $x_i \in \mathrm{b}_x[e]$, and e begins a biphrase, i.e. $\mathrm{d}[e] = 1$, otherwise it returns 0. The same operators also apply directly to triples, rather than edges. That is, if $t = i[e]$ is a triple, then $\mathrm{b}[t]$, $\mathrm{c}[t]$, $\mathrm{d}[t]$, $\mathrm{x}_i[t]$ and $\mathrm{y}[t]$ are defined as before.

We define the vocabulary of the lattice $G'(\mathbf{x})$ to be the set Σ of all enriched labels defined by the program in Figure 6.3. We know from the logic program that, given a pair of adjacent transitions (e, e') in a derivation from $G'(\mathbf{x})$, either: (1) they recognise adjacent target words within the same biphrase, i.e. $\mathrm{b}[e] = \mathrm{b}[e']$ and $\mathrm{d}[e'] = \mathrm{d}[e] + 1$; or (2) they represent the concatenation of two non-overlapping biphrases, i.e. $\mathrm{b}_x[e] \cap \mathrm{b}_x[e'] = \emptyset$ and $\mathrm{d}[e] = |\mathrm{b}_y[e]|$ (all words in $\mathrm{b}_y[e]$ have been recognised) and $\mathrm{d}[e'] = 1$ (e' begins the biphrase $\mathrm{b}[e']$).

Let a derivation $\mathbf{d} = \langle e_1, e_2, \dots, e_J \rangle$ be a sequence of J edges. The derivation recognises a string of triples $i[\mathbf{d}] = \langle i[e_1], i[e_2], \dots, i[e_J] \rangle = \langle t_1, t_2, \dots, t_J \rangle$, where each triple has the form $t_j = \langle b, d, c \rangle \in \Sigma$. We are also interested in the translation string defined by $\mathbf{d}$, that is, its target yield, which is denoted by $\mathrm{y}[\mathbf{d}] = \langle \mathrm{y}[e_1], \mathrm{y}[e_2], \dots, \mathrm{y}[e_J] \rangle = \langle \mathrm{y}[t_1], \mathrm{y}[t_2], \dots, \mathrm{y}[t_J] \rangle = \langle y_1, y_2, \dots, y_J \rangle$. Finally, let us also characterise $\mathbf{d}$ with respect to how it covers the input. Let $\mathrm{x}_i[\mathbf{d}]$ return the number of times the input word x_i is translated in $\mathbf{d}$, that is, $\mathrm{x}_i[\mathbf{d}] = \sum_{j=1}^{J} \mathrm{x}_i[e_j]$. A **valid derivation** is such that $\mathrm{x}_i[\mathbf{d}] = 1$ for

$i = 1 \dots I$. A valid derivation $\mathbf{d}$ may be accepted or rejected based on the ratio $r(\mathbf{d}) \equiv \frac{p(\mathbf{d})}{q^{(t-1)}(\mathbf{d})}$. If rejected, it may motivate a refinement that incorporates some additional context to the proposal bringing it closer to p. An **invalid derivation** is such that $\mathrm{x}_i[\mathbf{d}] \neq 1$ for some $i \in [1 \dots I]$. If $\mathbf{d}$ is an invalid derivation, then $\mathbf{d} \notin \mathcal{D}$, $p(\mathbf{d}) = 0$ and $r(\mathbf{d}) = 0$. Thus, an invalid derivation is always rejected and it may motivate a refinement that adds some of the non-overlapping constraints to the current proposal.

If a valid derivation $\mathbf{d}$ from $q^{(t-1)}$ motivates a refinement, we proceed as discussed in Section 4.2.3. That is, we identify in the yield $\mathbf{y} = \mathrm{y}[\mathbf{d}] = y_1^J$ a sequence $y_{j-k}y_{j-k+1} \dots y_j$ such that the knowledge of the n-gram y_{j-k+1}^j has been registered in $q^{(t-1)}$, but not that of the n-gram y_{j-k}^j. We then select the context y_{j-k+1}^{j-1} and extend it with one word to the left. The automaton $A^{(t)}$ is designed to update all 1-word continuations of this extended context y_{j-k}^{j-1}. That is, $A^{(t)}$ scales each occurrence of an n-gram of the form $y_{j-k}^{j-1}z$, where z is a target word, by a factor α_z as shown in Equation 6.1. We then obtain a refined proposal $q^{(t)} = q^{(t-1)} \cap A^{(t)}$ and bookkeep the n-grams $y_{j-k}^{j-1}z$ as described in Section 4.2.4.

$$\alpha_z = \frac{w_{k+1}(z|y_{j-k}^{j-1})}{w_k(z|y_{j-k+1}^{j-1})} \tag{6.1}$$

If an invalid derivation $\tilde{\mathbf{d}} = \langle e_1, e_2, \dots, e_J \rangle$ from $q^{(t-1)}$ motivates a refinement, then one possible strategy is to perform the update $q^{(t)} = q^{(t-1)} - D(\tilde{\mathbf{d}})$, where $D(\tilde{\mathbf{d}})$ is an unweighted deterministic automaton that accepts $\tilde{\mathbf{d}}$, and only $\tilde{\mathbf{d}}$.[14] The resulting proposal is such that

$$q^{(t)}(\mathbf{d}) = \begin{cases} q^{(t-1)}(\mathbf{d}) & \text{if } \mathbf{d} \neq \tilde{\mathbf{d}} \\ \bar{0} & \text{otherwise.} \end{cases}$$

[14]The difference operation between a wFSA A and an unweighted FSA B can be expressed in terms of the intersection between A and the complement of B. Because wFSAs are closed under complementation and under intersection (Hopcroft and Ullman, 1969), the difference operation returns another wFSA. Mohri (2009) presents a straightforward procedure for complementation of deterministic automata in linear time.

On the one hand, this operation is very cheap to compute and it increases the proposal by adding only J states on average. On the other hand, this update is extremely conservative incorporating non-overlapping constraints only in the specific context of $\tilde{\mathbf{d}}$. A much more extreme strategy would identify a position i for which $\mathrm{x}_i[\tilde{\mathbf{d}}] > 1$ and remove from $q^{(t-1)}$ every derivation $\mathbf{d}$ for which $\mathrm{x}_i[\mathbf{d}] > 1$. This can be done by (1) designing the automaton O_i (for "overlap at i") that recognises the language (over triples) $\Sigma^* X_i \Sigma^* X_i \Sigma^* \subseteq \Sigma^*$, where $X_i = \{t \in \Sigma : \mathrm{x}_i[t]\} \subseteq \Sigma$ is the set of triples from Σ that cover x_i; and (2) removing from $q^{(t-1)}$ all derivations compatible with O_i, i.e. $q^{(t)} = q^{(t-1)} - O_i$. Observe that paths in O_i contain at least two triples from X_i, therefore, at least two triples consuming i. This extreme strategy generalises from $\tilde{\mathbf{d}}$ preventing many more invalid derivations from ever happening. However, it does so at the cost of a large increase in the lattice associated with $q^{(t)}$. One can think of this strategy as incorporating to every state in $q^{(t-1)}$ one bit of information dedicated to encoding whether or not position i has been covered by any partial derivation reaching that state. This has the effect of multiplying the number of states in the lattice $q^{(t-1)}$ by two. In the extreme case, in which such constraints are added for ever input word, i.e. $i \in [1 \dots I]$, the resulting lattice enumerates all 2^I states necessary to perform inference in the absence of a distortion limit.[15]

Algorithm 11 illustrates the general procedure. In line 2, we start with a proposal $q^{(0)}$ that is the intersection between an upperbound on the translation lattice $G'(\mathbf{x})$ and an upperbound on the language model distribution $A^{(0)}$. Until convergence, we sample or optimise $\mathbf{d}$ from the current proposal (line 4). If $\mathbf{d}$ is invalid, then there is no need to evaluate $p(\mathbf{d})$ and we can set $r(\mathbf{d})$ to 0 (line 6). Otherwise, we compute the ratio between the goal and the proposal at $\mathbf{d}$ (line 8). In line 10, the derivation is accepted or rejected

[15]This extreme way of incorporating non-overlapping constraints is equivalent to what is done by Chang and Collins (2011) in the context of tightening a Lagrangian relaxation.

Algorithm 11 OS* for phrase-based SMT

```
1: $t \leftarrow 0$, converged $\leftarrow$ false                    ▷ in sampling also does $AR \leftarrow 0$
2: $q^{(0)} \leftarrow G'(\mathbf{x}) \cap A^{(0)}$
3: while not converged do
4:     $\mathbf{d} \leftarrow \text{search}(q^{(t)})$                              ▷ argmax or sample
5:     if $\exists i : \mathrm{x}_i[\mathbf{d}] \neq 1$ then                          ▷ if there are overlaps
6:         $r \leftarrow 0$
7:     else
8:         $r \leftarrow {p(\mathbf{d})}/{q^{(t)}(\mathbf{d})}$
9:     end if
10:    accept $\leftarrow$ assess$(r)$        ▷ deterministic in optimisation, random in sampling
11:    if not accept then                                        ▷ if $\mathbf{d}$ was rejected
12:        define $A^{(t+1)}$ based on $\mathbf{d}$ and $q^{(t)}$
13:        $q^{(t+1)} \leftarrow q^{(t)} \cap A^{(t+1)}$                                  ▷ update proposal
14:    end if
15:    $t \leftarrow t + 1$
16:    converged $\leftarrow$ update$(r, \text{accept})$                 ▷ in sampling this also updates $AR$
17: end while
18: return accepted samples along with $q^{(t)}$
```

depending on the ratio r and the mode in which OS* is running (optimisation or sampling). In optimisation, $\mathbf{d}$ is accepted if the ratio is close enough to one. In sampling, $\mathbf{d}$ is accepted with probability r. Note that, in either mode, if $r(\mathbf{d}) = 0$ the derivation is rejected. When a derivation is rejected we proceed by updating the proposal. If $\mathbf{d}$ is **valid**, in line 12, $A^{(t+1)}$ is an automaton that explicitly accounts for some k-gram context not yet registered in $q^{(t-1)}$. If $\mathbf{d}$ is **invalid**, in line 12, $A^{(t+1)}$ is an automaton that constraints $q^{(t-1)}$ by removing some overlapping derivations (including at least $\mathbf{d}$ itself).

Finally, to achieve faster intersections it is necessary to control the size of the lattice, which can be done by refining it, not everywhere, but rather in regions that are likely to participate in high-scoring derivations. The selective intersection introduced in the context of hierarchical models (see Section 5.3.2) can be used with phrase-based models in a straightforward manner. Moreover, it can be used with both types of refinements, namely, those that incorporate k-gram contexts, lowering the proposal, and those that remove invalid (overlapping) derivations from the proposal.

6.2.1 Relaxed lattices

We introduced an upperbound to $G(\mathbf{x})$ that is based on relaxing the non-overlapping constraint. In that relaxation, we replace the coverage vector (which prevents overlaps in Figure 6.2) by some simpler information, namely, a count of words translated and the most recently selected biphrase. Let us call that strategy TB, short for *tracking the last biphrase*. In this section we propose alternative upperbounds to $G(\mathbf{x})$ by relaxing the non-overlapping constraint in different ways. These alternative relaxations allow one to start from a space that already encodes a larger subset of the non-overlapping constraints, while avoiding $G'(\mathbf{x})$ to grow exponentially with the length of the input $\mathbf{x}$. The key is to allow fewer overlapping derivations in $G'(\mathbf{x})$ at the cost of some small (controlled) increase in space complexity.

The logic program in Figure 6.4 implements one such upperbound. Rather than using a bit vector to explicitly account for all non-overlap constraints, this program *tracks the last contiguous span* (TS) covered in the input, preventing biphrases to overlap with that span. An item in the program has the form $[\langle x_s^t \rangle, r, \gamma' y_j \bullet \gamma, c]$, where: (1) $\langle x_s^t \rangle$ represents the last contiguous input span covered; (2) r tracks the rightmost position covered in the input; (3) $\gamma' y_j \bullet \gamma$ is a dotted target phrase; and (4) is the number of source words covered thus far. Because we track the last contiguous sequence covered, rather than the last biphrase selected, we need an extra variable to track the last word translated, which is necessary for the computation of distortion costs.

The GOAL of the program is to prove items that cover exactly I words. The AXIOMS extend the empty hypothesis by selection of a biphrase. CONCATENATE extends an item $[\langle x_s^t \rangle, r, \gamma \bullet, c]$ by concatenation of a biphrase $\langle x_{s'}^{t'}, y_j \gamma' \rangle$ from left to right in target language order. The deduction only happens if the selected biphrase does not overlap with the span $\langle x_s^t \rangle$ and if the resulting item would cover at most I input words. The side condition $s' > t \vee t' < s$ prevents the overlap. If $\langle x_s^t \rangle$ and $\langle x_{s'}^{t'} \rangle$ are not adjacent, the

Goal $\quad [\langle x_s^t \rangle, r, \gamma \bullet, I]$

Axioms

$$\frac{R(x_s^t, y_j \gamma)}{[\langle x_s^t \rangle, t, y_j \bullet \gamma, t - s + 1] : \phi(\langle x_s^t, y_j \gamma \rangle) + \delta(0, s)}$$

Concatenate

$$\frac{[\langle x_s^t \rangle, r, \gamma \bullet, c] \; R(x_{s'}^{t'}, y_j \gamma')}{[\langle x_{s''}^{t''} \rangle, t', y_j \bullet \gamma', c + t' - s' + 1] : \phi(\langle x_{s'}^{t'}, y_j \gamma' \rangle) + \delta(r, s)} \quad \begin{array}{l} s' > t \vee t' < s \\ c + t' - s' + 1 \leq I \end{array}$$

where $s'' = s$ if $s' = t + 1$ else s'

$t'' = t$ if $t' = s - 1$ else t'

Scan

$$\frac{[\langle x_s^t \rangle, r, \gamma' \bullet y_j \gamma, c]}{[\langle x_s^t \rangle, r, \gamma' y_j \bullet \gamma, c] : \bar{1}}$$

Figure 6.4: TS: tracking the last contiguous input span covered.

last contiguous span in the consequent is that of the selected biphrase, i.e., $\langle x_{s''}^{t''} \rangle = \langle x_{s'}^{t'} \rangle$. If $\langle x_s^t \rangle$ and $\langle x_{s'}^{t'} \rangle$ are adjacent and the former precedes the latter in source language order, the last contiguous span in the consequent is $\langle x_s^{t'} \rangle$. If they are adjacent, but the latter precedes the former, then $\langle x_{s''}^{t''} \rangle = \langle x_{s'}^{t} \rangle$. Finally, Scan moves the dot forward in the target phrase, however, at this point the LM is not part of the parameterisation.

The program in Figure 6.4 is inspired by the relaxation in Chang and Collins (2011), who work in the context of exact optimisation using Lagrangian relaxation (see Section 3.3.2). However, it is important to highlight that Chang and Collins (2011) do make use of a distortion limit constraining the space of solutions reachable by their phrase-based model considerably — this is not the case in our approach. The program that implements Chang and Collins's relaxation exactly requires an additional condition to Concatenate which prunes out derivations that violate a distortion limit d, namely, $|r - s'| \leq d$.

The space complexity of the program is $O(I^7 B)$ where: (1) a linear fac-

GOAL $[\langle L, i, R\rangle, \gamma \bullet, I]$

AXIOMS

$$\frac{R(x_s^t, y_1\gamma)}{[\langle 0^{\min(d,s-1)}, t, 0^{\min(d,I-t)}\rangle, y_1 \bullet \gamma, t-s+1] : \phi(\langle x_s^t, y_1\gamma\rangle) + \delta(0,s)}$$

CONCATENATE

$$\frac{[\langle L, i, R\rangle, \gamma \bullet, c] \; R(x_s^t, y_j\gamma')}{[\langle L', t, R'\rangle, y_j \bullet \gamma', c+t-s+1] : \phi(\langle x_s^t, y_j\gamma'\rangle) + \delta(i,s)} \quad \begin{array}{l} c+t-s+1 \leq I \\ V \wedge V' = 0^I \end{array}$$

where $V = 0^{i-d-1}L1R0^{I-i-d}$ ▷ coverage vector of the antecedent
$V' = 0^{s-1}1^{t-s+1}0^{I-t}$ ▷ coverage vector of x_s^t
$b_1^I = V \vee V'$ ▷ updated coverage vector
$L' = \text{trim}(b_{t-d}^{t-1})$ ▷ d bits to the left of t
$R' = \text{trim}(b_{t+1}^{t+d})$ ▷ d bits to the right of t

SCAN

$$\frac{[\langle L, i, R\rangle, \gamma' \bullet y_j\gamma, c]}{[\langle L, i, R\rangle, \gamma' y_j \bullet \gamma, c] : \bar{1}}$$

Figure 6.5: TWW: tracking a window around the last word translated.

tor I is due to the need to track r; (2) another linear factor I is due to c; (3) a factor IB, where B is the maximum number of biphrases per source segment, is due to the dotted target phrases; (4) a quadratic factor I^2 is due to the longest contiguous span $\langle x_s^t\rangle$; and (5) another quadratic factor I^2 is due to the current biphrase. With a maximum phrase length L, (4) is lowered to IL^2. In relation to the program in Figure 6.3, this one produces fewer invalid derivations, particularly, when hypotheses are expanded covering contiguous input spans. In terms of complexity, it requires tracking $r \in [1 \ldots I]$ separately from $\langle x_s^t\rangle$.

The logic program in Figure 6.5 implements another upperbound to $G(\mathbf{x})$. This logic is an attempt to start from a search space that is closer to what a search space with a distortion limit looks like, however, still not imposing a distortion limit. Instead, the program prevents overlaps from happening only within a window of radius d around the last word translated. Beyond the limits of that window, overlaps might happen. Let us call this program TWW for *tracking window centred in the last word translated.* An item of the program has the form $[\langle L, i, R\rangle, \gamma' y_j \bullet \gamma, c]$ where: (1) $\langle L, i, R\rangle$ is a window of radius d around the last word translated i; L and R are coverage bit vectors of length d; they store coverage information for d positions to the left and to the right of i, respectively; (2) $\gamma' y_j \bullet \gamma$ is a dotted target phrase; and (3) c is the number of words translated thus far.

The GOAL of the program is to prove items covering exactly I words. The AXIOMS expand the empty hypothesis by selection of a biphrase. Observe how the last word x_t of the biphrase becomes the centre of the window and the coverage vectors start with at most d zeros each. CONCATENATE extends an item $[\langle L, i, R\rangle, \gamma \bullet, c]$ from left to right in target language order by concatenating the biphrase $\langle x_s^t, y_j\gamma'\rangle$ if x_s^t does not overlap with $\langle L, i, R\rangle$ and if the resulting item would cover at most I input words. Consider the vector $V = 0^{i-d-1}L1R0^{I-i-d}$ in the rule. V represents the coverage vector of the antecedent where everything before the first position represented by L and after the last position represented by R has been forgotten (completed by trailing zeros). The 1 in the middle represents the position i (last word translated) and $|V| = I$. The coverage vector $V' = 0^{s-1}1^{t-s+1}0^{I-t}$ represents the biphrase being selected, it covers exactly x_s^t. The side condition $V \wedge V' = 0^I$ prevents overlaps to happen within $\langle L, i, R\rangle$. The coverage information in the consequent is a trimmed down version of $b_1^I = V \vee V'$ (the vector updated via bitwise OR). $L' = \text{trim}(b_{t-d}^{t-1})$, where *trim* ignores positions below 1 and beyond I in the subsequence, is the part that concerns the left wing of the window, and similarly for R'. Finally, SCAN moves the dot forward in the target phrase without incorporating the LM weights at this point.

The space complexity of the program is $O(I^5 B 2^{2d})$ where: (1) a linear factor $2^{2d} I$ is due to the window $\langle L, i, R\rangle$, where the constant 2^{2d} is due to L and R; (2) a linear factor IB is due to the dotted target phrases; (3) another linear factor I is due to c; and (4) a quadratic factor I^2 is due to the number of segmentations of the input. With a maximum phrase length, (4) is made linear on input length.

It is straightforward to combine TS and TWW to track a window of radius d around the last contiguous input span. The logic in Figure 6.6 implements this *tracking window around the last contiguous span* (TWS) strategy. The program has items of the form $[\langle L, s, t, R\rangle, r, \gamma' y_j \bullet \gamma, c]$ where r is necessary to track the last word translated (for the purpose of computing distortion costs) and the pair (s, t) tracks the boundaries of the last contiguous span. The vectors L and R contain coverage information for d positions before s and d positions after t, respectively. The space complexity of this program is $O(I^7 B 2^{2d})$.

6.2.2 Dealing with overlapping phrases

In this section, we describe different strategies to deal with derivations containing overlapping phrases. In incorporating non-overlapping constraints, there is a trade-off between generalisation and representation cost. On the one hand, incorporating few constraints grows the lattice very little at each iteration, however, more iterations are required before convergence. Imagine the scenario in which each time an invalid derivation $\tilde{\mathbf{d}}$ is sampled (or optimised) we deal with it by using the conservative strategy introduced in Section 6.2, namely, we remove that specific derivation from the proposal. If our relaxation is such that many invalid derivations are possible in $G'(\mathbf{x})$, then they might keep on showing up as samples or optima and we will need to deal with them one by one. On the other hand, incorporating many constraints at once reduce the number of iterations necessary for convergence, but it largely increases the lattice. Exponentially large lattices become a problem for dynamic programming, which is in the core of most search algorithms including ours. The intersection with a wFSA that lowers the proposal

GOAL $[\langle L, s, t, R\rangle, r, \gamma\bullet, I]$

AXIOMS

$$\frac{R(x_s^t, y_1\gamma)}{[\langle L, s, t, R\rangle, t, y_1\bullet\gamma, t-s+1] : \phi(\langle x_s^t, y_1\gamma\rangle) + \delta(0, s)}$$

where $L = 0^{\min(d', s-1)}$
$R = 0^{\min(d'', I-t)}$

CONCATENATE

$$\frac{[\langle L, s, t, R\rangle, r, \gamma\bullet, c] \; R(x_{s'}^{t'}, y_j\gamma')}{[\langle L', s'', t'', R'\rangle, t', y_j\bullet\gamma', c+t'-s'+1] : w} \quad \begin{array}{l} c+t'-s'+1 \le I \\ (s' > t \vee t' < s) \\ V \wedge V' = 0^I \end{array}$$

where $V = 0^{s-d'-1}L1^{t-s+1}R0^{I-t-d''}$ ▷ coverage vector of the antecedent
$V' = 0^{s'-1}1^{t'-s'+1}0^{I-t'}$ ▷ coverage vector of x_s^t
$b_1^I = V \vee V'$ ▷ updated coverage vector
$s'' = s$ if $s' = t+1$ else s'
$t'' = t$ if $t' = s-1$ else t'
$L' = \mathrm{trim}(b_{s''-d}^{s''-1})$ ▷ d bits to the left of s”
$R' = \mathrm{trim}(b_{t''+1}^{t''+d})$ ▷ d bits to the right of t”
$w = \phi(\langle x_{s'}^{t'}, y_j\gamma'\rangle) + \delta(r, s')$

SCAN

$$\frac{[\langle L, s, t, R\rangle, r, \gamma'\bullet y_j\gamma, c]}{[\langle L, s, t, R\rangle, r, \gamma' y_j\bullet\gamma, c] : \bar{1}}$$

Figure 6.6: TWS: tracking the window around last contiguous span.

bringing it closer to the goal by incorporation of larger n-grams on demand is an example of a dynamic program. The Inside algorithm necessary for optimisation and sampling is another example.

If an invalid derivation $\tilde{\mathbf{d}} = \langle e_1, e_2, \ldots, e_J\rangle$ from $q^{(t-1)}$ motivates a refinement, one possible strategy already discussed (see beginning of Section 6.2) is to perform the update $q^{(t)} = q^{(t-1)} \cap \bar{D}(\tilde{\mathbf{d}})$. $D(\tilde{\mathbf{d}})$ is an unweighted deterministic automaton that accepts $\tilde{\mathbf{d}}$, and only $\tilde{\mathbf{d}}$, and $\bar{D}(\tilde{\mathbf{d}})$ its complement. The proposal $q^{(t)}$ will differ from $q^{(t-1)}$ only for the derivation $\tilde{\mathbf{d}}$. Let us call this strategy OD, a short for *one derivation*.

If OD represents one extreme, then AD (*all derivations*) represents the other. We perform the update $q^{(t)} = q^{(t-1)} \cap \bar{O}_i$ where O_i, is the automaton that recognises all derivations (without altering their weights) translating x_i multiple times and i is motivated by $\tilde{\mathbf{d}}$. The automaton $\bar{O}_i$ recognises the complement of the language $\Sigma^* X_i \Sigma^* X_i \Sigma^* \subseteq \Sigma^*$. The set $X_i = \{t \in \Sigma : \mathrm{x}_i[t] = 1\} \subseteq \Sigma$ contains all triples that begin a biphrase consuming x_i, i.e., $x_i \in \mathrm{b}_x[t] \wedge \mathrm{d}[t] = 1$.

Another extreme strategy somewhat related to AD is to *force the translation* of a certain word (FT). If some input word $x_{i'}$ is translated multiple times, then there exists at least one position i that was never translated, i.e., $\exists i : \mathrm{x}_i[\tilde{\mathbf{d}}] = 0$. This is true because every derivation must cover exactly I words. Let $M_i = \{t \in \Sigma : x_i \in \mathrm{b}_x[t]\} \subseteq \Sigma$ be the set of triples that cover the missing position. We can perform the update $q^{(t)} = q^{(t-1)} \cap F_i$, where F_i is the automaton that recognises the set of all derivations (without altering their weights) that necessarily translate i, i.e., $\Sigma^* M_i \Sigma^* \subseteq \Sigma^*$. Observe that this update does not enforce a non-overlapping constraint for position i, it simply requires i to be covered at least once. Both AD and FT implicitly add one bit to each state of the lattice to encode information about the position of interest duplicating the lattice in the worst case.

Many intermediary strategies can be designed. For example, one can remove all derivations overlapping at a certain position sharing a certain prefix in $\tilde{\mathbf{d}}$. Alternatively, if t_j and t_{j+k} are overlapping triples in $\tilde{\mathbf{d}}$, both consuming position i, one can remove derivations overlapping at i which share the substring $t_{j+|\mathrm{b}_y[t_j]|} \dots t_{j+k-1}$. Designing strategies to deal with overlapping phrases is a very experimental task and there is no net method. We discussed here some of the most straightforward. In what follows, we propose a more principled solution that capitalises on the current proposal to dynamically discover the set of constraints that are most often violated.

Suppose an invalid derivation is such that $\mathrm{x}_i[\tilde{\mathbf{d}}] > 1$. Also consider we are working with the strategy AD. Thus, we proceed by computing the set $X_i = \{t \in \Sigma : \mathrm{x}_i[t] = 1\} \subseteq \Sigma$ of all triples consuming x_i. We can rely on edge marginal weights to have a summarised view of X_i that contains only the most likely triples X'_i. Alternatively, one could make use of sampling at any moment, regardless of the mode in which OS* performs, to explore the distribution finding a certain $X'_i \subseteq X_i$ which contains triples likely to overlap due to x_i. Finally, we chose O'_i the automaton that recognises the language $\Sigma^* X'_i \Sigma^* X'_i \Sigma^* \subseteq \Sigma^*$ and perform the update $q^{(t)} = q^{(t-1)} \cap \bar{O}'_i$. This will add a relevant set of non-overlapping constraints at a lower cost than the original AD strategy. This strategy can be seen as a form of **selective intersection**, where rather than implicitly incorporating an additional bit of information to every state of the lattice $q^{(t-1)}$, we implicitly do that only in the context of triples that are likely to be sampled from $q^{(t-1)}$.

A final direction that we have left for future work concerns the use of Lagrangian Relaxation (LR) (see Section 3.3.2). In LR, some of the hard constraints associated with a problem can be modelled through weights (Lagrangian multipliers). These multipliers are easy to add to the parameterisation of the model and they serve the purpose of rewarding (or penalising) features that capture certain aspects of the constraints to be modelled. Unlike actual hard constraints, these features are designed to be local to each of the steps in a derivation. Thus, these multipliers do not increase the search space complexity. In phrase-based SMT, these local features encode how many times each position is translated, and the multipliers reward/penalise each biphrase on the basis of the positions it covers. We can think of LR as a technique that allows hard constraints to be softly represented by weights. Chang and Collins (2011) have successfully applied Lagrangian relaxation to account for a large subset of the non-overlapping constraints necessary to perform exact decoding (optimisation) with phrase-based models. However, they work under the assumption of a distortion limit, and they do not address

GOAL $[\langle x_i^{i'}, \text{EOS} \bullet \rangle, I]$

AXIOMS

$$\frac{R(x_i^{i'}, y_1\gamma)}{[\langle x_i^{i'}, y_1 \bullet \gamma\rangle, i'-i+1] : \phi(\langle x_i^{i'}, y_1\gamma\rangle) + \delta(0,i) + w_2(y_1|\text{BOS})}$$

CONCATENATE

$$\frac{[\langle x_l^{l'}, \gamma' y_{j-1} \bullet \rangle, c] \;\; R(x_i^{i'}, y_j\gamma)}{[\langle x_i^{i'}, y_j \bullet \gamma\rangle, c+i'-i+1] : \phi(\langle x_i^{i'}, y_j\gamma\rangle) + \delta(l',i) + w_2(y_j|y_{j-1})} \quad \begin{array}{l}(i > l' \vee i' < l)\\ c+i'-i+1 \leq I\end{array}$$

SCAN

$$\frac{[\langle x_i^{i'}, \gamma' y_{j-1} \bullet y_j\gamma\rangle, c]}{[\langle x_i^{i'}, \gamma' y_{j-1}y_j \bullet \gamma\rangle, c] : w_2(y_j|y_{j-1})}$$

ACCEPT

$$\frac{[\langle x_i^{i'}, \gamma y_j \bullet \rangle, I]}{[\langle x_i^{i'}, \text{EOS} \bullet \rangle, c] : w_2(\text{EOS}|y_j)}$$

Figure 6.7: Initial proposal starting from a 2-gram LM

sampling.

6.3 Initial proposal

As we discussed in Section 6.2.1, the program in Figure 6.3 implements an upperbound $G'(\mathbf{x})$ to $G(\mathbf{x})$ by relaxation of the non-overlapping constraint. The program creates a translation lattice where adjacent choices of biphrases never overlap (see the side condition in CONCATENATE). While in Sections 6.2 and 6.2.1 we omitted the LM from the parameterisation, the program in Figure 6.7 shows that it is easy to incorporate a low-order upperbound on the LM distribution. Particularly, it is straightforward to start from a bigram representation, rather than a unigram representation (used in the hierarchical case). Because each state in the lattice defined by the program implicitly encodes information about a specific target phrase, it is straightforward to add a component $w_2(y_j|y_{j-1})$ that upperbounds $p_{lm}(y_j|y_{j-1})$ (see Section 4.2.1) without further increasing the space complexity.

The logic in Figure 6.7 implicitly initialises a proposal $q^{(0)} = G'(\mathbf{x}) \cap A^{(0)}$, where $G'(\mathbf{x})$ is chosen to be the derivations produced by the program in Figure 6.3 and $A^{(0)}$ is chosen to be an optimistic 2-gram version of the full language model A. In practice, the logic produces $q^{(0)}$ in one go without the need to explicitly compute the intersection and without the need to explicitly instantiate $A^{(0)}$. However, it is necessary to bookkeep the target bigrams compatible with $q^{(0)}$, which is straightforward to do at the time of its creation.

The same argument applies to the other relaxations discussed in Section 6.2.1.

6.4 Updating the proposal

In this section we describe how a proposal can be refined by the incorporation of additional n-grams and how it can be constrained by the addition of non-overlapping constraints. Both operations are defined in terms of standard weighted finite-state *intersection*. Constraints can also be added to lattices by means of the *difference* operation, otherwise seen as a special case of intersection which requires *complementation*.

The logic program in Figure 6.8 implements the weighted finite-state intersection (Mohri et al., 1996; Cohen et al., 2008). The input of the program is a pair of wFSAs, namely, $\langle \Sigma, Q_1, \langle I_1, \lambda_1 \rangle, \langle F_1, \rho_1 \rangle, \langle E_1, \omega_1 \rangle \rangle$ and $\langle \Sigma, Q_2, \langle I_2, \lambda_2 \rangle, \langle F_2, \rho_2 \rangle, \langle E_2, \omega_2 \rangle \rangle$ (see Section 2.2.1). In our application, the first automaton is acyclic and represents a proposal distribution $q^{(t-1)}$, the second automaton is not necessarily acyclic and represents some form of refinement $A^{(t)}$. Both automata are defined over a vocabulary Σ of triples of the kind $\langle b, d, c \rangle$ (see Section 6.2). The output is an acyclic wFSA that represents a refined distribution $q^{(t)} = q^{(t-1)} \cap A^{(t)}$ (see Section 2.1.3.1) and has

$$\textsc{Goal} \qquad [v \in F]$$

$$\textsc{Axioms}$$
$$\frac{}{[\langle q_1, q_2\rangle] : \lambda_1(q_1) \otimes \lambda_2(q_2)} \quad q_1 \in I_1 \wedge q_2 \in I_2$$

$$\textsc{Move}$$
$$\frac{[\langle q_1, q_2\rangle] \; E_1(q_1 \xrightarrow{t_1:w_1} r_1) \; E_2(q_2 \xrightarrow{t_2:w_2} r_2)}{[\langle r_1, r_2\rangle] : w_1 \otimes w_2} \quad t_1 = t_2$$

$$\textsc{Accept}$$
$$\frac{[\langle q_1, q_2\rangle]}{[v_g] : \rho_1(q_1) \otimes \rho_2(q_2)} \quad q_1 \in F_1 \wedge q_2 \in F_2$$

Figure 6.8: wFSA intersection

the form $q^{(t)} = \langle \Sigma, Q \subseteq Q_1 \times Q_2, \langle I = \{v_i\}, \lambda\rangle, \langle F = \{v_g\}, \rho\rangle, \langle E, \omega\rangle\rangle$. For convenience, $q^{(t)}$ has a single initial node v_i and a single goal node v_g.

An item of the program is a pair of states (one from A_1 and another from A_2). It has the form $[\langle q_1, q_2\rangle]$ where $q_1 \in Q_1$, $q_2 \in Q_2$ and $\langle q_1, q_2\rangle$ is associated with a state in the output wFSA. An edge $q \xrightarrow{t:w} r$, also denoted by an edge item $E(q \xrightarrow{t:w} r)$, is such that it specifies a transition from q to r with weight w and labelled with a triple $t \in \Sigma$. An edge item $E(q \xrightarrow{t:w} r)$ is a short for $e = \langle q, t, r\rangle \in E$ with weight $w = \omega[e]$.

The Goal of the program is to prove the goal node v_g. The Axioms combine the initial states of A_1 and A_2 via the crossproduct $I_1 \times I_2$, it also incorporates their weights. Move combines two edges (e_1, e_2), one from each automaton, to move from state $\langle q_1, q_2\rangle$ to state $\langle r_1, r_2\rangle$ where $(t[e_1] = q_1, t[e_2] = q_2)$ and $(h[e_1] = r_1, h[e_2] = r_2)$. It also incorporates the $\otimes$-product of the weights of the edges. The side condition states that the edges must be labelled with the same triples, i.e., $t_1 = t_2$ where $t_1 = i[e_1]$ and $t_2 = i[e_2]$. Finally, Accept incorporates the weights of final nodes from F_1 and F_2 under a single goal node v_g.

6.4.1 LM refinement

Suppose $\mathbf{d} = \langle e_1, e_2, \ldots, e_J \rangle$ a valid derivation from $q^{(t-1)}$ that has been rejected. In Section 6.2, it was explained that $\mathbf{d}$ motivates an update in the proposal $q^{(t-1)}$ which extends n-grams whose context is the sequence y_{j-k+1}^{j-1} with one word to its left, where y_{j-k}^{j-1} is a sequence in $\mathbf{y} = \mathrm{y}[\mathbf{d}]$. The vocabulary of the proposal distribution $q^{(t-1)}$ is made of triples, not of target words. Therefore, the automaton that performs the update must be specified in terms of triples.

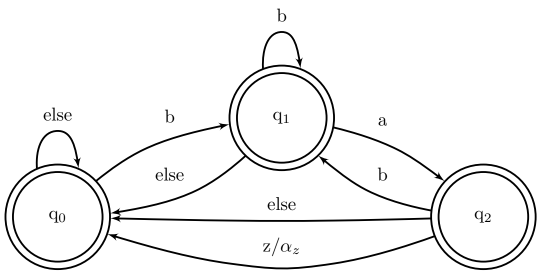


Figure 6.9: Down-weighting z in the context of ba

To motivate the idea let us consider a simple update that incorporates the trigram baz. From Section 4.2.3, the automaton that performs the update is defined as illustrated in Figure 6.9, where $\alpha_z = \frac{w_3(baz)}{w_2(az)}$. Consider the set of all triples that produce a certain target word y, i.e., $\Sigma_y = \{t \in \Sigma : \mathrm{y}[t] = y\} \subseteq \Sigma$. A simple strategy that is directly compatible with the algorithm in Figure 6.8 is to design $A^{(t)}$ to reweight sequences compatible with the language $\Sigma_b \Sigma_a \Sigma$. That is, if a transition is to recognise the target word y from state q to state r, then $A^{(t)}$ contains $|\Sigma_y|$ transitions from q to r, each one labelled with a triple in Σ_y. This means, that in the automaton shown in Figure 6.9 there would be a set of transitions from 0 to 1 recognising the triples in Σ_b, each of which would be weighted by $\bar{1}$. The keyword *else* in state 0 would then refer to the set of triples that do not yield b, i.e., $\Sigma_{\bar{b}} = \{t \in \Sigma : \mathrm{y}[t] \neq b\}$. A transition $q_2 \xrightarrow{a:\alpha_a} q_0$ would be replaced by $|\Sigma_a|$ transitions, each of which

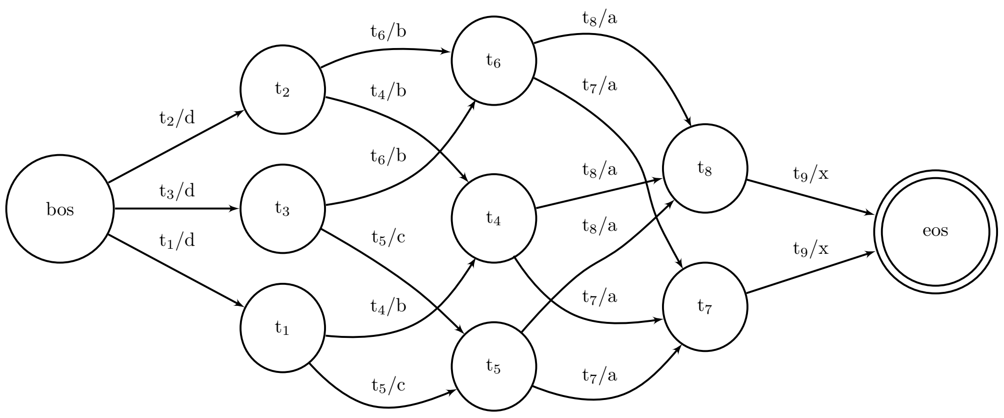


Figure 6.10: Example of proposal

would recognise a symbol from Σ_a and would carry a weight α.

Consider the example in Figure 6.10 where edges are annotated with triples and, for the sake of illustration, the letter after the forward slash makes explicit the yield of the triple. The refinement that updates all sequences of three triples such that the first two triples project onto the target string ba is illustrated in Figure 6.11. In the figure, $t_i \in \Sigma$ is a triple, $\Sigma_b = \{t_4, t_6\}$ are triples whose yield is b, and $\Sigma_a = \{t_7, t_8\}$ are triples whose yield is a. Transitions from q_2 to q_0 are weighted by $\alpha_i = \frac{w_3(y_i|ba)}{w_2(y_i|a)}$ where $y_i = \text{y}[t_i]$ is the target word produced by the triple. This strategy can be directly applied to longer n-grams.

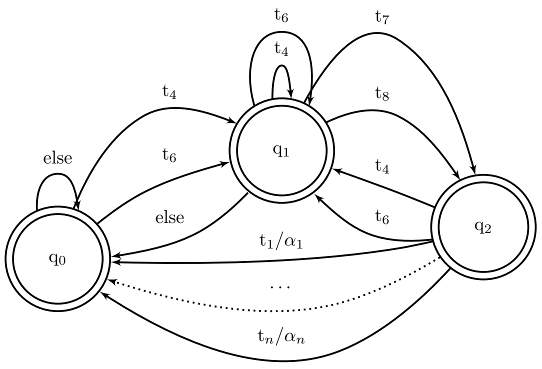


Figure 6.11: Down-weighting $\Sigma_b\Sigma_a\Sigma$

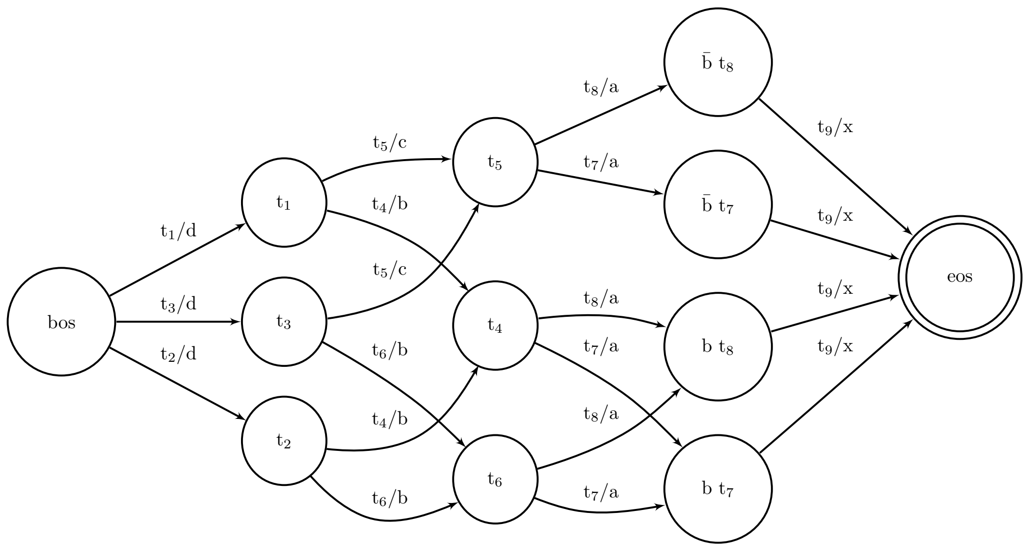


Figure 6.12: Example of refined proposal

The intersection between $q^{(t-1)}$, as illustrated in Figure 6.10, and $A^{(t)}$, as illustrated in Figure 6.11, is shown in Figure 6.12. Observe how the intersection separates *everywhere in the lattice* paths that include the substring *ba* from those that do not. This might expand the lattice in regions that have low probability.

It is interesting to observe that a refinement could also be defined in terms of the very specific sequence of triples observed in a derivation. Suppose again that **d** motivates a refinement that updates the weight of all n-grams of the kind $y_{j-k}^{j-1}z$ where z is a target word. The sequence y_{j-k}^{j-1} is associated with a sequence of triples t_{j-k}^{j-1} in $i[\mathbf{d}] = \langle t_1, t_2, \ldots, t_J \rangle$. A refinement that updates 1-triple continuations of t_{j-k}^{j-1}, i.e. $t_{j-k}^{j-1}u$ where $u \in \Sigma$ is a triple, relates to a very specific region of the lattice $q^{(t-1)}$, shared by fewer derivations than those that are compatible with the string y_{j-k}^{j-1}. Therefore, such a refinement would cause a more modest growth of the lattice.

Suppose $i[\mathbf{d}] = \langle t_1, t_4, t_7, t_9 \rangle$ and **d** a derivation from the proposal in Figure 6.10, besides, $\mathrm{y}[\mathbf{d}] = \langle d, b, a, x \rangle$. Let us assume that *bax* motivates a refinement, that is, $w_3(x|ba) < w_2(x|a)$. The automaton in Figure 6.13

illustrates a refinement that down-weights derivations compatible with the language $t_4t_7\Sigma$, that is, all 1-triple continuations of t_4t_7. In the figure, $t_i \in \Sigma$ is a triple. Transitions from q_2 to q_0 carry a weight $\alpha_i = \frac{w_3(y_i|ba)}{w_2(y_i|a)}$, where $y_i = \mathrm{y}[t_i]$ and $ba = \mathrm{y}[t_4]\mathrm{y}[t_5]$.

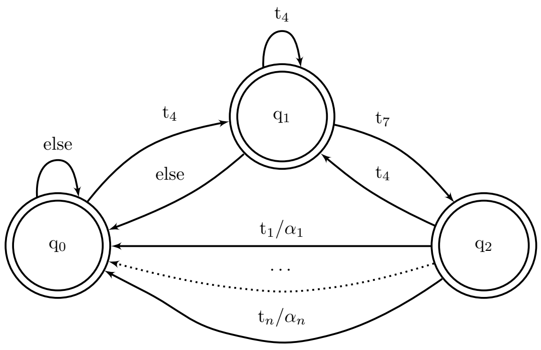


Figure 6.13: Down-weighting $t_4t_7\Sigma$

In this case, the bookkeeping is done in terms of the vocabulary of triples, rather than the vocabulary of target words. In comparison to what was done for hierarchical models, the first strategy, based on n-grams of target words, is similar to refining on terminal types, while the second strategy, based on n-grams of triples, is similar to refining on terminal instances (see Section 5.3.2).

Finally, edge marginals can be used to group triples that yield the same target word on the basis of how likely they are to participate in a derivation sampled or optimised from the current proposal distribution. This is a straightforward application of what was developed for hierarchical models in terms of selective intersection (see Section 5.3.2). Using the Inside-Outside algorithm we can compute the marginal weight $\mathsf{M}[e]$ of each edge $e \in E$ in the lattice. Let $E_t = \{e \in E : i[e] = t\}$ be the set of edges that are labelled with a given triple t. The total weight that the lattice gives to a triple t is given by $\mathsf{M}[t] = \sum_{e \in E_t} \mathsf{M}[e]$, where by abuse of language we call $\mathsf{M}[t]$ the marginal weight of a triple. For a certain refinement that reweights the language $\Sigma_b\Sigma_a\Sigma$, such as exemplified earlier, we can select a subset of $\Sigma_b\Sigma_a$ according to how much weight the lattice gives to triples in Σ_b and Σ_a.

6.4.2 Adding constraints

A constraint is typically designed as an unweighted deterministic automaton $A^{(t)}$ that recognises a language specifying overlapping triples. For example, $\Sigma^* X_i \Sigma^* X_i \Sigma^*$ is the language that recognises all strings of triples that overlap at position i. In such cases, it is necessary to remove derivations from $q^{(t-1)}$ that are accepted by $A^{(t)}$.

The difference $q^{(t-1)} - A^{(t)}$ can be performed in terms of the intersection $q^{(t-1)} \cap \bar{A}^{(t)}$ where $\bar{A}^{(t)}$ is the automaton that accepts the complement of the language $L(A^{(t)})$. Complementation is a simple operation done in time linear with the size of $A^{(t)}$. For epsilon-free automata of the kind we use, it requires two simple steps (Mohri, 2009):

completion $A^{(t)}$ must be made complete, that is, it must accept Σ^*.

1. make a copy of $A^{(t)}$ and call it $\bar{A}^{(t)}$;
2. augment it with an additional state q' with self-loops labelled with all symbols in Σ;
3. for every state q originally in $A^{(t)}$, add transitions of the kind $q \xrightarrow{t} q'$ for all symbols $t \in \Sigma$ that are not yet associated with a transition from q.

complementation the final states of $\bar{A}^{(t)}$ are made non-final and vice-versa (q' is therefore made final). Finally, all weights are set to $\bar{1}$ so that $q^{(t-1)} \cap \bar{A}^{(t)}$ does not alter the weight of derivations incompatible with $A^{(t)}$.

6.5 Conditioning context

At the time $G'(\mathbf{x})$ is created, one can easily store the set B of all target bigrams that participate in derivations in $G'(\mathbf{x})$. The logic in Figure 6.7 proves this point, however, in the context of integrating an optimistic bigram LM component. This set B can be given to Algorithm 10, prior to intersection with $A^{(0)}$, to further tighten the LM upperbound distribution (see Section 4.2.2.2). The intuition is that we know the specific 1-word contexts in which a target word z may be yielded by a derivation in $G'(\mathbf{x})$. This information limits the possible context histories in the computation of the LM upperbound distribution resulting in a smaller gap between $q^{(0)}$ and p. Integrating this tighter 2-gram LM component does not require an explicit intersection, neither it requires the explicit instantiation of $A^{(0)}$. Instead, we can update the weight of each transition in the relaxed lattice in time proportional to the number of transitions in $G'(\mathbf{x})$. The resulting automaton is equivalent to $q^{(0)}$ produced by the program in Section 6.3. The same can be done for the other relaxations discussed in Section 6.2.1.

6.6 Experiments

For the experiments reported in this section we used the French-English portion of the 6th version of the Europarl collection (Koehn, 2005). In all experiments, we used the Moses toolkit (Koehn et al., 2007) to extract a phrase table following Koehn et al. (2003).[16] Language models were trained by *lmplz* (Heafield et al., 2013) using the English monolingual data made available by the WMT (Callison-Burch et al., 2012). That is, *Europarl*, *newscommentaries* and *news-2012*. The models extracted from different monolingual corpora were combined via interpolation using *newstest2010* as the development set. In all experiments, parameter estimation was performed via

[16]We extracted phrase-tables containing at most 10 translations per source phrase.

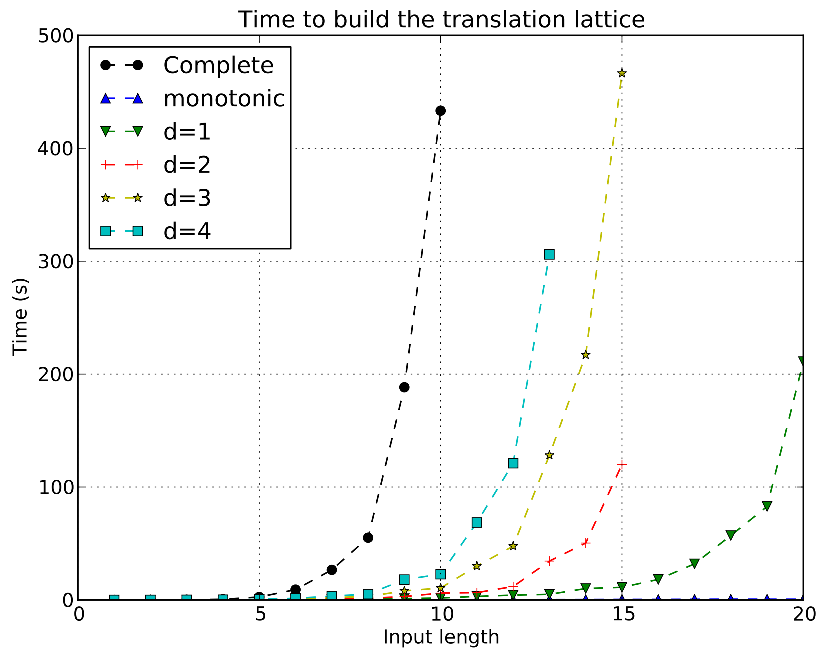


Figure 6.14: Time to build a translation lattice $G(\mathbf{x})$ that encodes all of the non-overlapping constraints

MERT on the *newstest2010* development set using single reference BLEU as the oracle. Finally, from the *newstest2011* test set, we selected sentences randomly according to their length. We sampled 20 examples for each class of length from 1 to 30 words.

Let us start by considering the time necessary to build a translation lattice $G(\mathbf{x})$ that includes all of the non-overlapping constraints. That is, without any form of relaxation. Figure 6.14 shows the average time in seconds as a function of the input length. In the figure, we compare different types of lattices. A complete lattice with no limit on distortion (reordering) is represented by the curve marked with circles. The curve marked with an up triangle represents monotonic translation. The remaining curves represent non-monotonic translation up to a maximum distortion limit d from 1 to 4. The number of non-overlapping constraints grows exponentially with

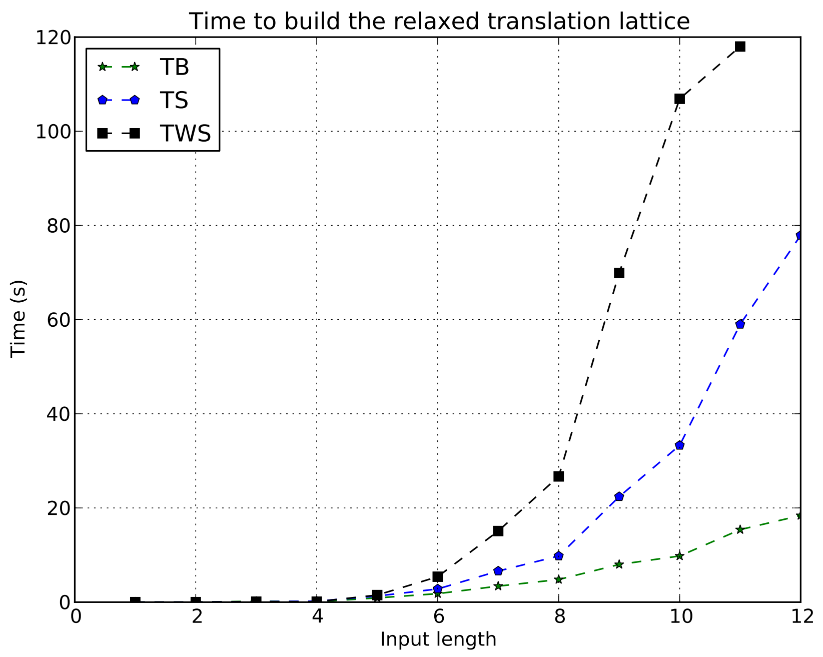


Figure 6.15: Time necessary to build a relaxed translation lattice as a function of the input length for different types of relaxation.

the length of the input, hence, the steep curves in Figure 6.14 (except for monotonic translation).

We can make use of the relaxations discussed in Section 6.2.1 to lower the complexity of building the translation lattice. Figure 6.15 compares different strategies in terms of their average running times (also shown in Figure 6.16 using logarithmic axes). As expected, relaxing the lattices leads to more reasonable running times even though we operate with no limits in reordering. Comparing the different relaxations, we see that incorporating additional constraints upfront requires longer running times. However, as discussed in Section 6.2.1, the increase is polynomial with the input length rather than exponential. The more constrained lattices (TS and TWS) are also bigger, as Figure 6.17 shows.

It turns out that the cheap computation of the relaxed translation lattices

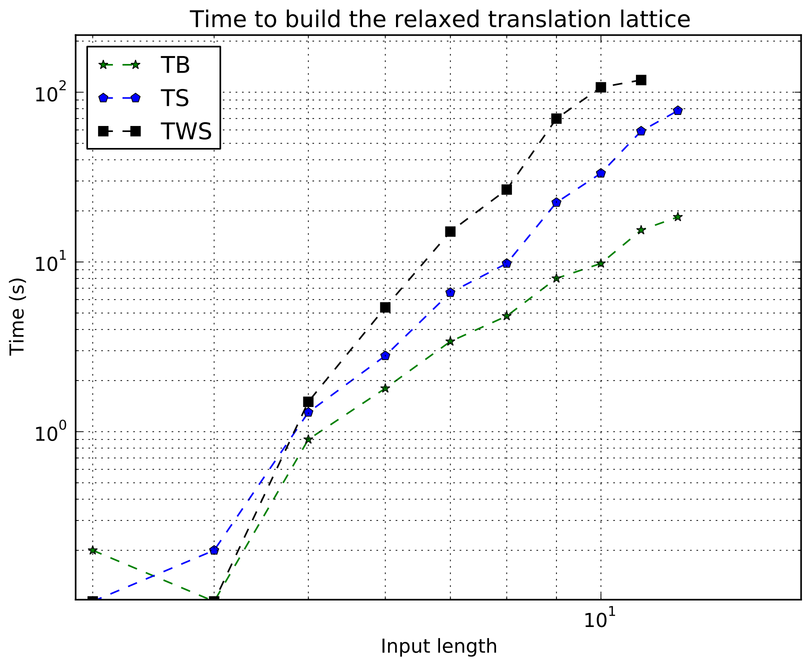


Figure 6.16: Time to build a relaxed lattice. Both axes are shown in logarithmic scale to simplify analysis of complexity.

comes at a high cost. Most of the search effort necessary for convergence to an optimum or to a minimum acceptance rate level after that will be focused on removing invalid derivations from the lattice. That is, most of the samples (in either modes of OS*) are **invalid** due to overlapping derivations. In the absence of valid samples OS* cannot gather evidence to support LM refinements, which means that the algorithm has to spend a lot of time constraining the lattice. To illustrate how severe this problem is, consider the curves in Figure 6.18. They show the number of refinements of the type $\bar{O}_i$ necessary to achieve a minimum of 10% valid (non-overlapping) derivations in a sample of 10 thousand derivations drawn from the proposal. Recall that a refinement of this type remove all derivations overlapping at position i. It is implicitly equivalent to incorporating one extra bit of information to every state in the lattice, hence, it multiplies the size of the lattice by two on

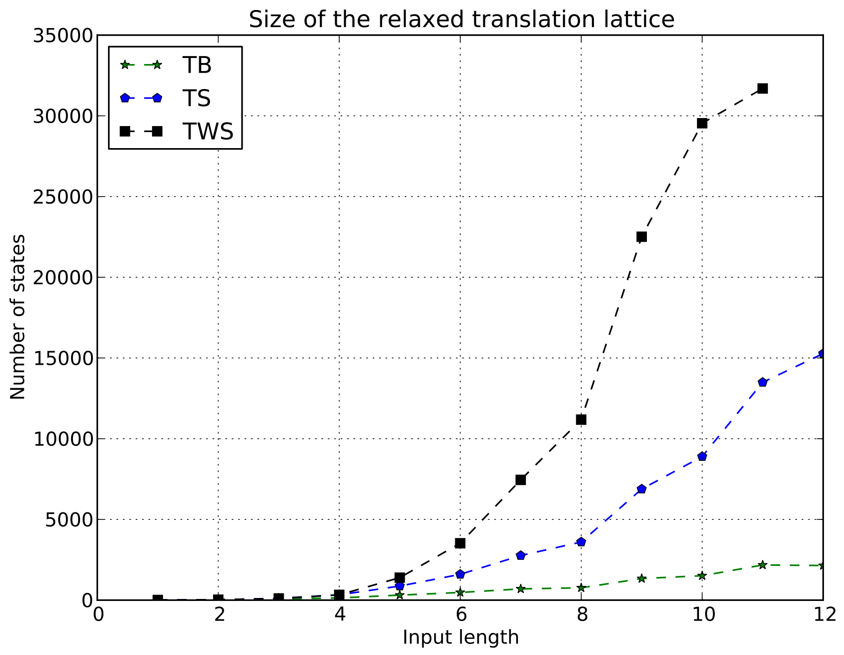


Figure 6.17: Initial size of translation lattices produced by different algorithms

average. After each refinement, we extract a new batch of samples and re-evaluate the rate of valid derivations. Note that this rate is an upperbound to the acceptance rate, since invalid samples are automatically rejected. Figure 6.18 shows that to achieve at least 10% of valid derivations using $\bar{O}_i$ refinements, we need to completely prevent overlaps at most of the positions (all but 2 on average). This means we have to start close to a complete lattice, which is impracticable for long sentences in the absence of a distortion limit.

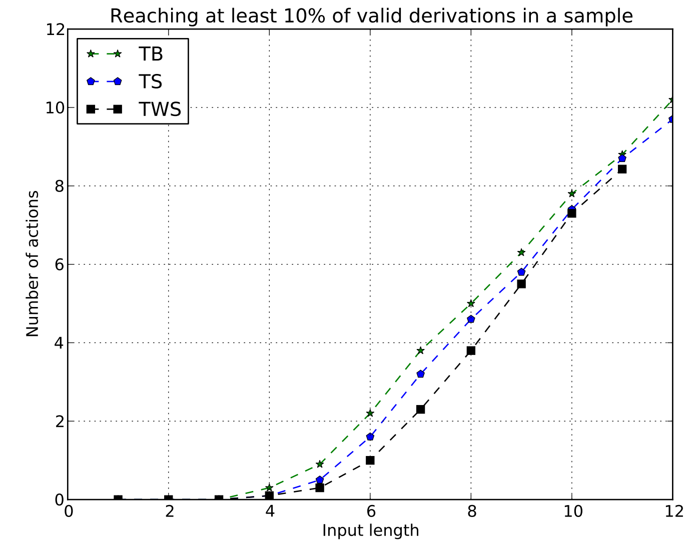


Figure 6.18: Number of $\bar{O}_i$ refinements necessary to achieve at least 10% of valid derivations in a sample of 1 thousand derivations drawn from the proposal

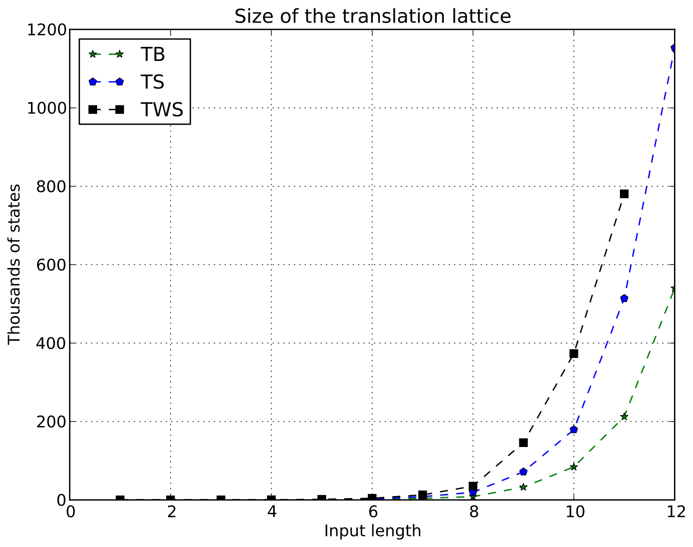


Figure 6.19: Size of proposal lattices constrained up to a 90% OV rate

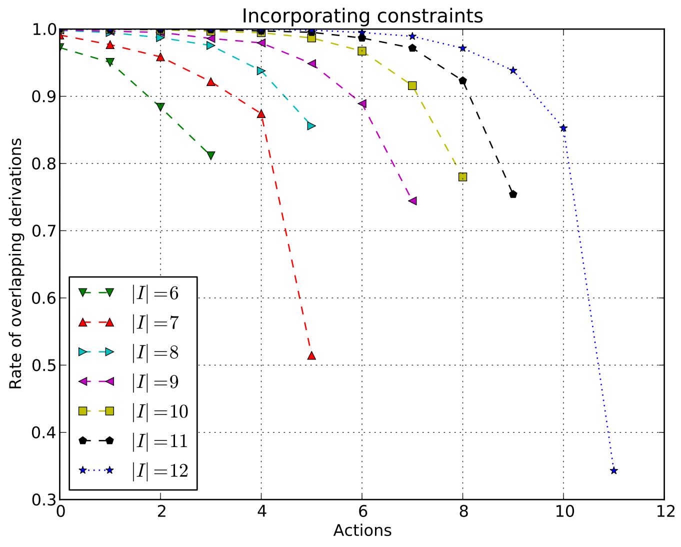


Figure 6.20: Progress of the OV rate as a function of the number of $\bar{O}_i$ refinements upon a proposal initially built with the algorithm TB

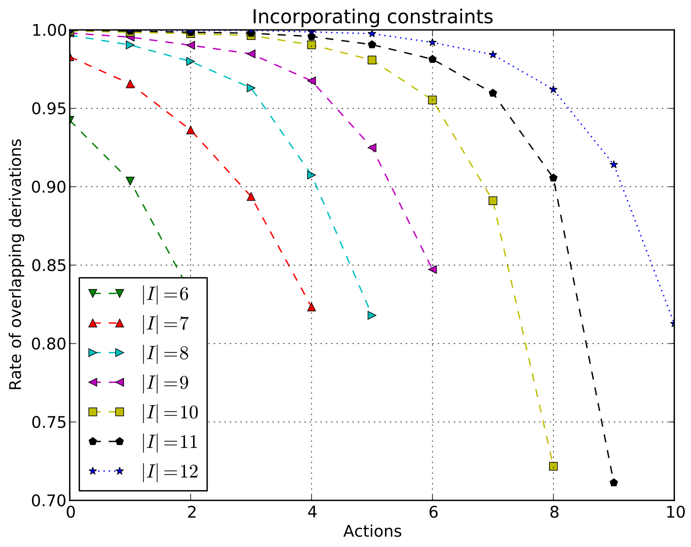


Figure 6.21: Progress of the OV rate as a function of the number of $\bar{O}_i$ refinements upon a proposal initially built with the algorithm TS

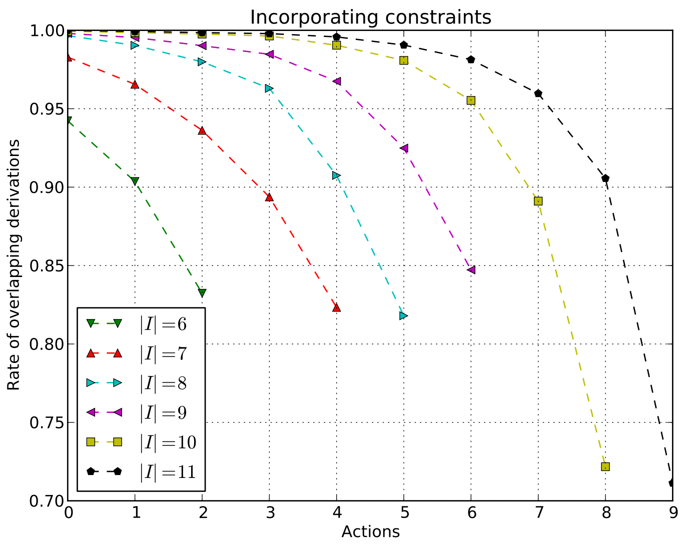


Figure 6.22: Progress of the OV rate as a function of the number of $\bar{O}_i$ refinements upon a proposal initially built with the algorithm TWS

Figures 6.20, 6.21 and 6.22 detail the rate of invalid derivations (OV rate) from proposals built using TB, TS and TWS, respectively. We can see that it takes many constraints before the OV rate starts dropping. It is important to highlight that refinements of the type $\bar{O}_i$ cannot distinguish triples according to how likely they are. Therefore, they might still be wasteful in that less aggressive operations could be applied to most of the input positions without implying an exponential growth of the lattice. Finally, Figure 6.19 shows the final size of the lattice after the OV rate is dropped below 90% (i.e. after the upperbound on the acceptance rate is equal or superior to 10%).

To confirm whether or not $\bar{O}_i$ is wasteful, at least two strategies are possible. One would be to remove invalid paths as they are optimised or sampled from the proposal. However, this turns out to be too time-inefficient. The algorithm spends too many iterations removing invalid paths one by one instead of progressing with LM refinements. Another strategy relies on sam-

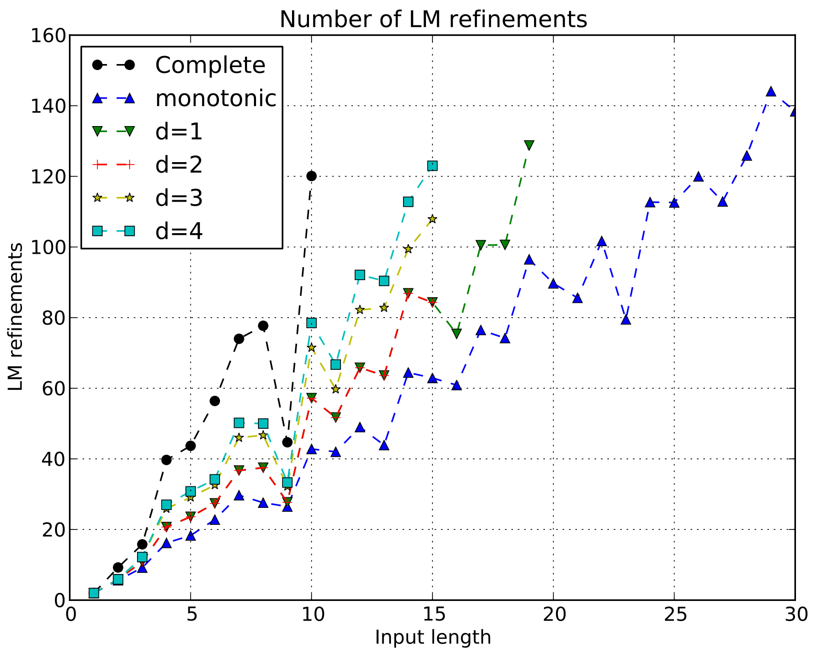


Figure 6.23: Number of LM refinements before convergence to the true optimum with a 3-gram LM. The initial proposals are not relaxed with respect to the non-overlapping constraints, that is, they do not contain invalid derivations.

pling to produce a summary of the distribution from which we gather more information about overlapping triples. We perform sampling with respect to L^1 identifying from each batch of samples the pairs of triples that overlap the most. We then design an operation to remove all derivations overlapping due to those triples. This strategy is applied iteratively until the OV rate is lowered below 90%. It turns out that this strategy is not fast enough, neither it produces smaller automata than $\bar{O}_i$ refinements do. To explore to which extent the growth of the automata is due to the incorporation of constraints in regions of the lattice with low probability, we also performed sampling relative to L^α with $\alpha \in [1, 10]$. The intuition in trying larger values of α is to gather more samples from regions of the distribution with more probability mass. Once more, the procedure was not faster, neither more economic than

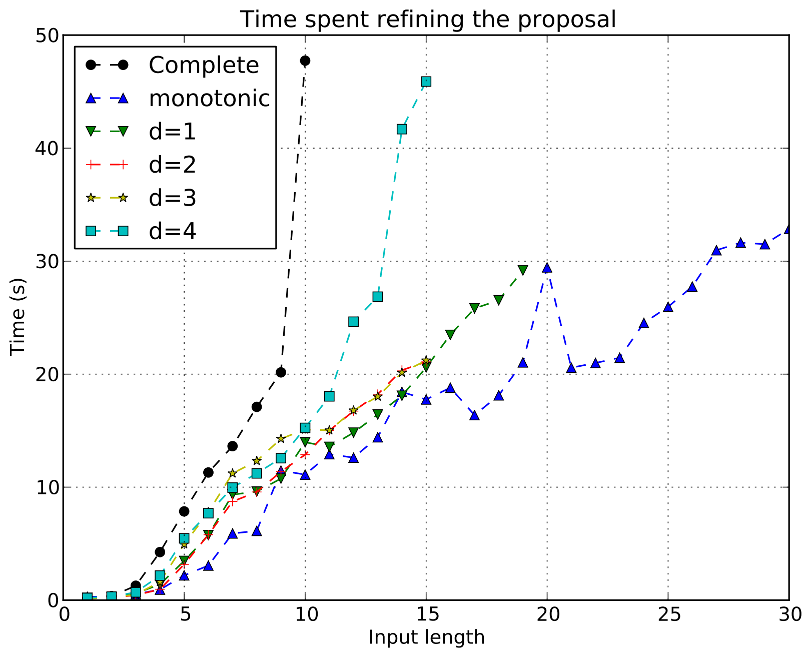


Figure 6.24: Time necessary to refine a fully-constrained lattice up to convergence to the true optimum with a 3-gram LM.

$\bar{O}_i$, confirming that there are too many invalid paths which score highly. In face of this inefficiency, in the following, we show the performance of OS* only with respect to LM refinements. To do that, we have to start from fully-constrained proposals.

We start by inspecting the efficiency of OS* with respect to LM refinements in optimisation mode. Figure 6.23 shows the number of refinements on top of the fully-constrained lattices shown in Figure 6.14. It is very encouraging to see that the number of refinements appear to grow linearly with the length of the input. Moreover, allowing wider reordering does not seem to change this dependency. The conclusions are similar with respect to time efficiency, shown in Figure 6.24. In Figure 6.25, the axes are converted to a logarithm scale (a *loglog* plot) to simplify the analysis of the complexity.

At all times during the search, provided that we have obtained at least

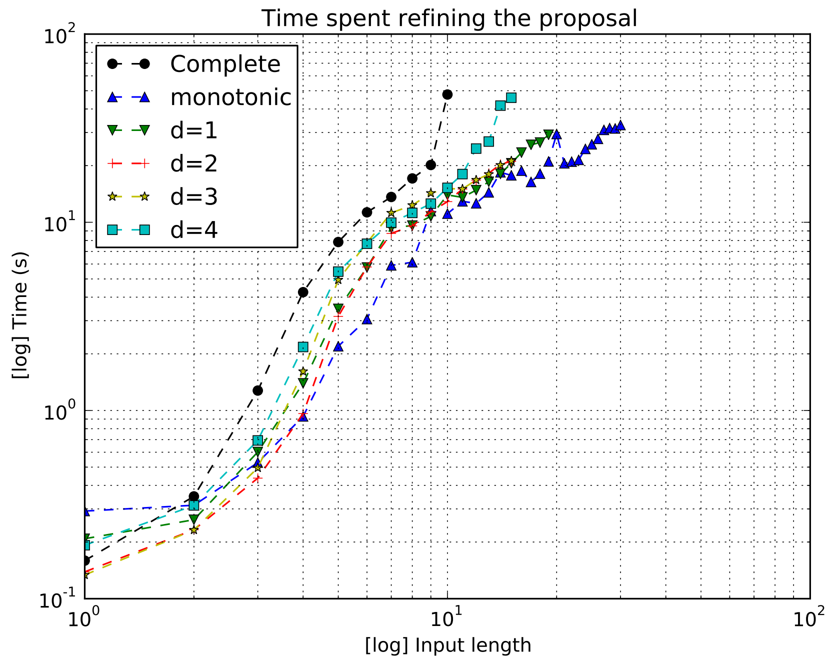


Figure 6.25: Time necessary to refine a fully-constrained lattice up to convergence to the true optimum with a 3-gram LM. Both axes are shown in logarithmic scale to illustrate how the distortion limit does not increase the dependency of the decoding time on input length.

one **valid** derivation, we have a lowerbound on the score of the true optimum ($\mathbf{d}^*$). If we think about all the valid derivations optimised from q at any given moment, there will be one that has the best score so far according to p. Let us denote that derivation by $\mathbf{d}'$. Because q is an upperbound to p, we know that the score of the true optimum derivation must lie between $p(\mathbf{d}')$ (the true score of the best derivation found thus far) and $\max_{\mathbf{d}} q(\mathbf{d})$ (the maximum of q). That is, $p(\mathbf{d}') \leq p(\mathbf{d}^*) \leq \max_{\mathbf{d}} q(\mathbf{d})$. This observation allows for an error-safe space optimisation of the lattice based on edge marginals. In a nutshell, the idea is to remove from q edges whose marginal weight (computed under the *max-sum* semiring via application of the Inside-Outside algorithm) is worse than the $\max_{\mathbf{d}} q(\mathbf{d})$ by a factor β.[17] Our application of

[17] Goodman (1998) proposes this pruning strategy in the context of probabilistic parsing.

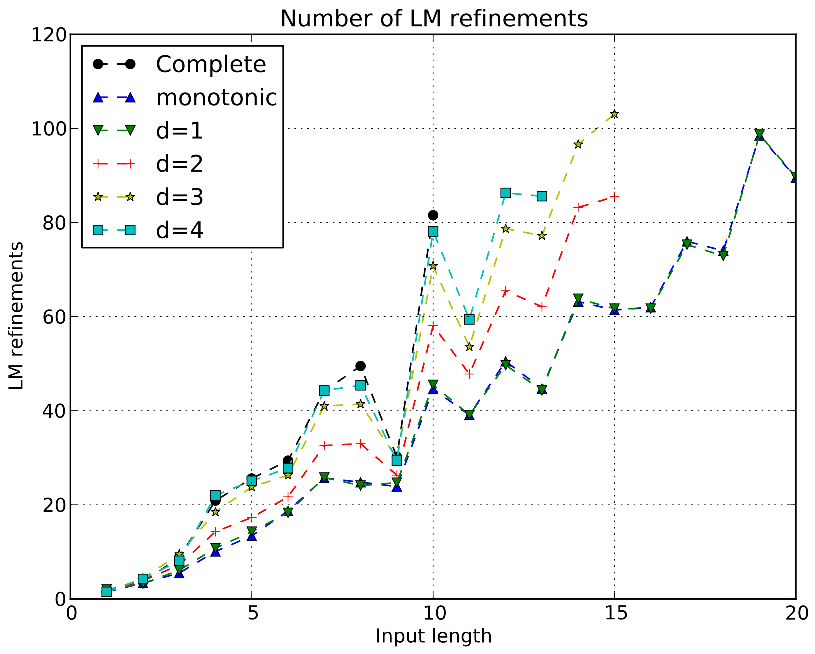


Figure 6.26: Number of LM refinements necessary to achieve 15% of acceptance rate with a 3-gram LM. The initial proposals are not relaxed with respect to the non-overlapping constraints, that is, they do not contain invalid derivations.

this technique is error-safe because we have an upperbound on how far from the true optimum we are, this upperbound is given by $\max_{\mathbf{d}} q(\mathbf{d}) - p(\mathbf{d}')$. Setting $\beta = \max_{\mathbf{d}} q(\mathbf{d}) - p(\mathbf{d}')$ means that we can never prune the optimum away, therefore the space optimisation will not be prone to search errors. We reserve the use of this technique to optimisation only, since in sampling we cannot guarantee exactness if the marginals are not preserved.

The following concerns results with OS* running in sampling mode.[18]

The algorithm has been also presented – either verbatim or slightly modified – under different names in other works (Sixtus and Ortmanns, 1999; Charniak and Johnson, 2005; Huang, 2008; Dyer, 2010). Sixtus and Ortmanns (1999) call it *forward-backward pruning*. Graehl (2005) calls it *relatively-useless pruning*. Huang (2008) calls it *forest pruning*. Dyer (2010) calls it *max-marginal* pruning to emphasise the use of the *max-sum* semiring.

[18]Currently we use MERT to train the model's weight vector, which is normalised by its L_1 norm in the Moses implementation. While optimisation is not sensitive to the scale

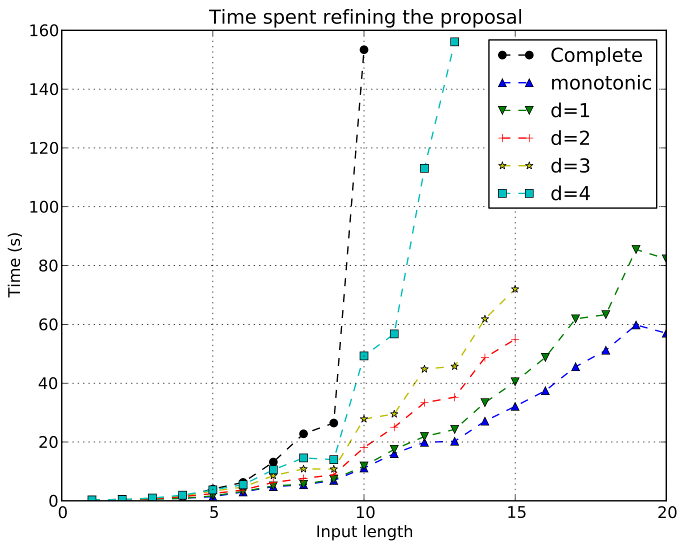


Figure 6.27: Time necessary to refine a fully-constrained lattice up to a 15% acceptance rate with a 3-gram LM.

Figure 6.26 shows the number of refinements and Figure 6.27 shows the convergence time to achieve 15% of acceptance rate starting from the fully-constrained lattices shown in Figure 6.14. Again, the number of refinements appear to grow linearly with the length of the input and allowing wider reordering does not seem to change this dependency. The time figures for sampling are slightly worse than in the optimisation case mostly due to the fact that we cannot apply error-safe pruning. Therefore the lattices are always growing at every iteration. In future work, we intend to asses our samplers with respect to appropriate decision rules, such as MBR decoding. Samples also should lead to better parameter estimation, a direction that is left for future work.

of the weights, in sampling the scale determines how flat or peaked the distribution is. (Arun et al., 2010) experiment with scaling MERT-trained weights as to maximise BLEU on held-out data, as well as with MBR training. A more adequate training algorithm along similar lines is reserved for future work.

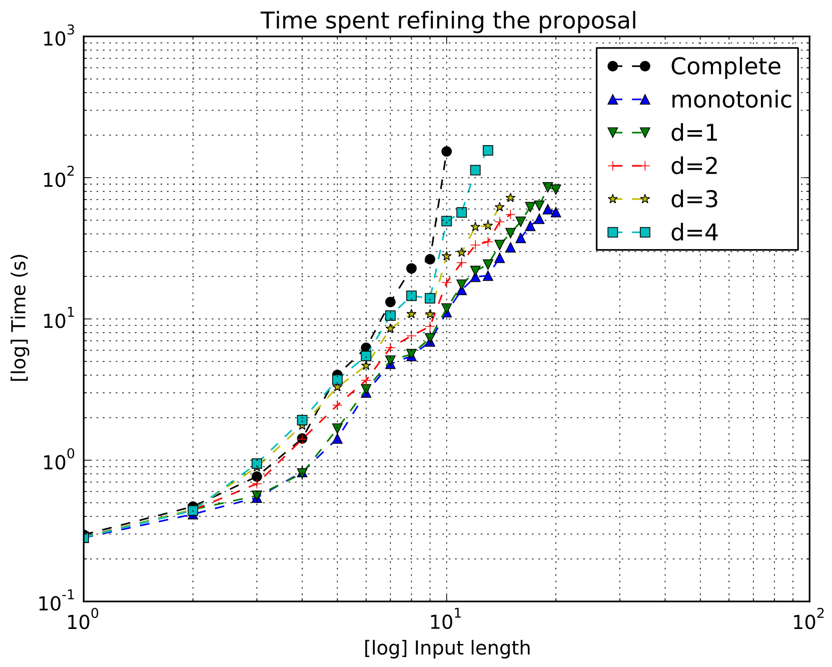


Figure 6.28: Time necessary to refine a fully-constrained lattice up to a 15% acceptance rate with a 3-gram LM. Both axes are shown in logarithmic scale to illustrate how the distortion limit does not increase the dependency on input length.

Chapter 7

Conclusions

In SMT, a model of translational equivalences explains all correspondences between a source and a target language. In hierarchical phrase-based SMT, translational equivalences are represented by a synchronous context-free grammar. The rules in the grammar compactly encode translation operators including long-distance reordering phenomena. In phrase-based SMT, translational equivalences are represented by a cascade of finite-state transducers. In such models reordering is arbitrary and typically softly constrained by features, rather than hard-coded in the set of transformation rules. A parameterisation of the model defines an unnormalised distribution over translation derivations. This is the goal distribution over which we wish to perform inference.

The parameterisation is typically defined in terms of a log-linear combination of features that independently capture different aspects of the quality of the bilingual correspondences and a language model component which captures the well-formedness of the translation. On the one hand, the bilingual correspondences are typically assessed by features that are local to each step in a derivation (e.g. deciding on the translation of a phrase). On the other hand, the language model component is a form of nonlocal parameterisation which depends on the interaction between multiple steps in a derivation. Nonlocal parameterisation violates structural independence assumptions complicating the underlying distribution which in turn means that a fully parameterised model is typically intractable even for short sentences with a bigram language model.

In a more abstract way, the space of translation derivations of a certain input text can be seen as the intersection between a locally parameterised translation hypergraph and the target language model distribution. The translation hypergraph compactly represents the space of weighted translation derivations compatible with the input under the model of translational equivalences. The intersection with the language model effectively re-scores these derivations accounting for target language fluency. In hierarchical SMT, the translation hypergraph is a tractable and acyclic wCFG, also referred to as a translation forest. In phrase-based SMT, the translation hypergraph is a generally intractable and acyclic wFST, also referred to as a translation lattice. However, under the condition of hard limits in reordering, this translation lattice can be made tractable. The target language model typically assumes an nth order conditional independence assumption and because of that it can be represented as a recursive wFSA whose size is exponential with the order of the language model.

Inference in SMT is typically done in terms of optimisation. That is, one typically searches for the highest-scoring translation derivation under the fully parameterised model, a task known as decoding. Due to the complexity of the underlying search space, decoding is typically done with algorithms that heuristically approximate the space defined by the intersection between the translation hypergraph and the target language model. Beam-search and cube-pruning are examples of such approximations. They construct only part of the space of translation derivations guided by heuristics that attempt to predict high-scoring derivations while low-scoring ones are pruned. They have been shown to scale well to long sentences and to produce translations of reasonable accuracy. However, they lack formal guarantees and they do not support sampling.

This thesis focused on the problem of performing exact inference in terms of sampling and optimisation in phrase-based and hierarchical phrase-based SMT. Our approach is based on lowering the complexity associated with the incorporation of nonlocal parameterisation. More specifically, lowering the computational complexity of the intersection between the translation hypergraph and the target language model. To achieve that we employ the OS* algorithm which unifies optimisation and sampling in a single framework based on a cross between adaptive rejection sampling and A* optimisation. In sum, we avoid explicitly computing the complete intersection by instead computing a sequence of tractable proposal hypergraphs which all upper-bound the intractable goal distribution. These proposals are simpler hypergraphs which "forget" the context of n-grams. We use the OS* technique to refine these hypergraphs by incrementally intersecting longer n-grams on the basis of evidence of the need to do so. Evidence is gathered by optimising and sampling directly from these proxy distributions by means of standard dynamic programming techniques.

The approach we have presented is, to the best of our knowledge, the first one to address the problem of exact sampling for machine translation and to do that in a framework that also handles exact optimisation. The contributions of this thesis included studying the extent to which it is computationally wasteful to fully incorporate nonlocal parameterisation to the structure of a translation hypergraph. We focused on the incorporation of an n-gram language model component to illustrate a form of nonlocal parameterisation. The language model was an appealing choice due to its importance towards generating quality translations. In the case of phrase-based SMT we proposed to formulate the non-overlapping constraint as a form of structural dependency that complicates the search space and that can be overlooked at first to be later incorporated on demand.

Let us review the hypotheses of this thesis:

H1 a full intersection with the language model is computationally wasteful;

H2 explicitly incorporating n-grams and constraints can be done rather locally in regions of the hypergraph that are associated with high-scoring (or likely) derivations;

H3 in the context of phrase-based SMT, we can overlook the non-overlapping constraints at first and introduce them on demand;

H4 rejection sampling can help lower the complexity associated with non-local parameterisation by effectively sampling from a simpler model (with more local dependencies) and assessing the samples with a more complex model (with less local dependencies);

Our experiments show that only a fraction of the language model n-grams need to be incorporated to the translation hypergraph (both in hierarchical and in phrase-based SMT) in order to perform exact inference, confirming **H1**. Results on selective intersection confirm **H2** in the context of incorporating an n-gram language model component. We have shown that, by intersecting n-grams only with edges whose marginals are the highest-scoring, we can achieve convergence in fewer iterations taking less time and explicitly expanding fewer states of the hypergraphs. In the context of phrase-based SMT, incorporating non-overlapping constraints on demand did not turn out to be sufficiently efficient. The relaxations to the translation lattice introduce an exponential number of invalid derivations. We have found that many of these derivations in fact score highly enough to considerably slow down the progress of the search. This means **H3** is only true for very short sentences and will not hold in the general case. There has been some evidence in recent related work that non-overlapping constraints can be dealt with via soft constraints in the form of Lagrangian multipliers, a direction which we intend to pursue in future work. Finally, in the context of sampling, our experiments show that there is potential in performing rejection sampling with a distribution of lower complexity (e.g. a model which incorporates a 2-gram LM component) to obtain exact samples from a distribution of higher complexity (e.g. a model which incorporates a 4-gram LM component), supporting **H4**. We intend to explore this direction in more detail in future work.

We have also shown the importance of tightening our upperbounds on the LM distribution on the basis of the active vocabulary of the translation hypergraphs. For that we also introduced efficient algorithms. We also introduced and tested a variant of the Earley procedure which achieves a consistent speed up by capitalising on the incremental aspects of the LM refinements we rely on.

At this point, due to issues with scalability we have been limited to experimenting with short sentences. Because of that, we have left for future work a more qualitative analysis of the translations produced by our technique. While our technique does not make search errors, this improvement might not directly reflect in improved translation quality unless parameter estimation is done in the context of our exact optima and unbiased samples. The current issues with scalability prevent us from performing more suitable parameter estimation at this point and therefore we have been experimenting with models trained via MERT using beam-search and cube-pruning.

Finally, the most important conclusion is that nonlocal parameterisation of the kind of the language model component can be incorporated on demand. This opens up the possibility of working with dynamic programming in unpruned hypergraphs, therefore achieving exact optimisation and unbiased sampling. Unbiased samples are of great use and have been very little explored in SMT. Moreover, our framework gives other forms of strong guarantees. For instance, in optimisation at all times we know how far from the optimum we can be in the worst case. In sampling, we have access to an unbiased estimate of the acceptance rate. These indicators can be used to better understand the search errors made by popular heuristics and they can also be used to early stop the search with some measure of the degree of approximation we achieved. There are many directions in which we could improve our methodology and there are other applications that could benefit from our results. In the following we discuss some of these directions.

Future work

Our approach can be improved in a number of directions, including the following:

Lagrangian relaxation: in the context of phrase-based SMT, we could use of Lagrangian relaxation to implicitly incorporate non-overlapping constraints by penalising popular overlaps on the basis of Lagrangian multipliers. Incorporating Lagrangian relaxation to our setup would require some modifications to the general Lagrangian relaxation framework, but it would remain feasible and compatible with OS*. In a nutshell, we would need to modify the proposals to account for a vector of multipliers μ. This vector contains one multiplier for each position of the input text. We would modify the proposals we optimise and sample from as follows: $q_\mu(\mathbf{d}) \equiv q(\mathbf{d}) - \mu \mathrm{v}[\mathbf{d}]$. The first part $q(\mathbf{d})$ is the standard upperbound introduced in this thesis, namely, $G(\mathbf{x}) \cap A^{(0)}$. The second part is a penalty on the number of violations of the non-overlapping constraint. The violations are assessed for each input position and captured by $\mathrm{v}_i[\mathbf{d}] \equiv \mathrm{x}_i[\mathbf{d}] - 1$, where $i \in [1 \ldots I]$ are positions of the input and $\mathrm{x}_i[\mathbf{d}]$ returns the number of times the ith input word has been translated in $\mathbf{d}$. Observe that q_μ remains an upperbound to p. If $\mathbf{d}$ is a valid derivation, than $\mathrm{v}[\mathbf{d}] = 0$ and $q_\mu(\mathbf{d}) = q(\mathbf{d})$. If $\mathbf{d}$ is an invalid derivation, than $\mathrm{v}[\mathbf{d}] \neq 0$ and $p(\mathbf{d}) = -\infty$ (i.e., p is undefined for invalid derivations). The component $\mu \mathrm{v}[\mathbf{d}]$ can be incorporated directly to the parameterisation without violating structural independence, that is, they represent a form of local parameterisation that does not increase the complexity of the translation lattice. Therefore, the framework would be compatible with both OS* optimisation and OS* sampling. Some additional modifications to the subgradient descent algorithm that optimises the Lagrangian multipliers might also be necessary. The intuition is to implicitly incorporate as many of the constraints as possible via adjusting the multipliers. Only then, if it is still necessary to lower a gap in optimisation, or if the acceptance rate stopped improving below the desired level due to a high rate of invalid derivation, we resort to refinements that explicitly constrain the lattice.

General speed-ups: we could make use of error-safe pruning based on edge marginals and the error bound given by OS* in our hierarchical decoder. We also want to investigate similar techniques that would be more compatible with sampling. However, the case of sampling is less straightforward because we have to make sure the marginals are consistent with the goal distribution. Moreover, if we start pruning edges we will loose guarantees. Another potentially important speed-up is to reimplement some heavily iterative procedures (e.g. intersection) in a programming language such as C++. Our current prototype is implemented in python (an interpreted language).

Improved upperbounds: our strategies of selective intersection have parameters that influence convergence speed and depend on various aspects of the distribution, such as the length of the input, how many refinements have already been done at a certain point of the search, the order of the LM refinement, etc. We want to investigate techniques to adaptively change these parameters in the course of the search based on the progression of the upperbound. Another direction is to make use of sampling to summarise the distribution and decide on explicitly incorporating some of the n-grams upfront starting from a tighter proxy distribution q. We can also optimise the LM refinements (a small recursive automaton) with respect to how much they lower the upperbound. Currently, they focus only on the 1-word continuations of a k-gram context. That is, the upperbound weight of substrings of those n-grams are left unchanged, whereas we could take take the opportunity to tighten the upperbound distribution further.

Parameter estimation: once convergence time is improved we can contribute towards performing better parameter estimation. For example, we can perform minimum risk training using exact samples. Exact samples should give a much more diverse view that the narrow one given by n-best list approximations. Another important aspect is that sampling is sensitive to the scaling of the model parameters, therefore a learning algorithm that is meant to operate in the framework of sampling becomes important. Moreover, with our method we can have an unbiased estimate of the true partition function, that is, the L^1 norm of the goal distribution, which is essential for training. This estimate can be seen as the expectation of the partition function of the tractable proxy distribution with respect to the distribution $r(\mathbf{d}) = \frac{p(\mathbf{d})}{q(\mathbf{d})}$. This expectation can be estimated by pointwise approximations or on the basis of a batch of samples and its value evolves in the course of the search as q is lowered. Finally, we can compute an upperbound on the variance of the estimate, which we can also tighten as the search progresses.

Qualitative analysis: once convergence time is improved we can compare exact optimisation, exact sampling and approximate algorithms in terms of the quality of the translations they produce. We can also explore more interesting decision rules such as MBR decoding and analyse the impact of more suitable parameter estimation based on sampling.

Nonlocal parameterisation: we intend to explore the idea of using OS* to achieve a sufficiently efficient proxy q for a model p_1 and then use that proxy as an upperbound to a more complex model p_2 with even more complex nonlocal parameterisation. We could then rely on standard rejection sampling where p_2 is the goal and q is the upperbound. A direct application of this idea would be to incorporate features that require a complete derivation to be assessed. One example is a Probabilistic Weighted Context-Free Grammar (PCFG) LM, that is, a syntax-aware language model distribution. The intersection between a translation hypergraph and a PCFG LM is a computationally expensive procedure, which is only closed under specific conditions. However, it might be the case that the PCFG LM would not drastically change the distribution p, but rather modulate it ("change it slightly"). If this is the case, we could expect that the decrease in the acceptance rate of a sampler designed for a standard wFSA LM will be small enough for rejection sampling to remain feasible with a PCFG LM. Other features can account for other forms of analysis such as word-sense, topic distribution, lexical consistency within a document, etc. This framework could allow the incorporation of features that depend on a much more global context. Intuitively, these features should contribute to finer aspects of quality.

Bibliography

Aho, A. V. and Ullman, J. D. (1969). Syntax directed translations and the pushdown assembler. *Journal of Computer and System Sciences*, 3(1):37–56.

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., and Mohri, M. (2007). Openfst: a general and efficient weighted finite-state transducer library. In *Proceedings of the 12th international conference on Implementation and application of automata*, CIAA'07, pages 11–23, Berlin, Heidelberg. Springer-Verlag.

Arun, A., Dyer, C., Haddow, B., Blunsom, P., Lopez, A., and Koehn, P. (2009). Monte carlo inference and maximization for phrase-based translation. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, CoNLL '09, pages 102–110, Stroudsburg, PA, USA. Association for Computational Linguistics.

Arun, A., Haddow, B., Koehn, P., Lopez, A., Dyer, C., and Blunsom, P. (2010). Monte carlo techniques for phrase-based translation. *Machine Translation*, 24(2):103–121.

Aziz, W., Dymetman, M., and Venkatapathy, S. (2013). Investigations in exact inference for hierarchical translation. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 472–483, Sofia, Bulgaria. Association for Computational Linguistics.

Baker, J. K. (1979). Trainable grammars for speech recognition. In *Proceedings of the Spring Conference of the Acoustical Society of America*, pages 547–550, Boston, MA.

Bar-Hillel, Y., Perles, M. A., and Shamir, E. (1961). On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, (14):143–172.

Baum, L. E., Petrie, T., Soules, G., and Weiss, N. (1970). A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *The Annals of Mathematical Statistics*, 41(1):164–171.

Berger, A. L., Pietra, S. A. D., Brown, P. F., Pietra, V. J. D., Gillett, J. R., Kehler, A. S., and Mercer, R. L. (1996a). Language translation apparatus and method of using context-based translation models.

Berger, A. L., Pietra, V. J. D., and Pietra, S. A. D. (1996b). A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Bickel, P. J. and Doksum, K. A. (1977). *Mathematical Statistics: Basic Ideas and Selected topics.*, volume I. Prentice-Hall, Inc., Oakland, CA, USA, 2nd edition.

Billot, S. and Lang, B. (1989). The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 143–151, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Blunsom, P., Cohn, T., Dyer, C., and Osborne, M. (2009). A gibbs sampler for phrasal synchronous grammar induction. 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*, pages 782–790, Suntec, Singapore. Association for Computational Linguistics.

Blunsom, P., Cohn, T., and Osborne, M. (2008). A discriminative latent variable model for statistical machine translation. In *Proceedings of ACL-08: HLT*, pages 200–208, Columbus, Ohio. Association for Computational Linguistics.

Blunsom, P. and Osborne, M. (2008). Probabilistic inference for machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 215–223, Stroudsburg, PA, USA. Association for Computational Linguistics.

Brown, P., Cocke, J., Pietra, S. D., Pietra, V. D., Jelinek, F., Lai, J., and Mercer, R. (1995). Method and system for natural language translation.

Brown, P. F., Cocke, J., Pietra, S. A. D., Pietra, V. J. D., Jelinek, F., Lafferty, J. D., Mercer, R. L., and Roossin, P. S. (1990). A statistical approach to machine translation. *Computational Linguistics*, 16:79–85.

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 19:263–311.

Callison-Burch, C., Koehn, P., Monz, C., Post, M., Soricut, R., and Specia, L. (2012). Findings of the 2012 workshop on statistical machine translation. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 10–51, Montréal, Canada. Association for Computational Linguistics.

Caraballo, S. A. and Charniak, E. (1998). New figures of merit for best-first probabilistic chart parsing. *Comput. Linguist.*, 24(2):275–298.

Carter, S., Dymetman, M., and Bouchard, G. (2012). Exact Sampling and Decoding in High-Order Hidden Markov Models. In *Proceedings of the*

2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 1125–1134, Jeju Island, Korea. Association for Computational Linguistics.

Chang, R. and Hancock, J. (1966). On receiver structures for channels having memory. *IEEE Trans. Inf. Theor.*, 12(4):463–468.

Chang, Y.-W. and Collins, M. (2011). Exact decoding of phrase-based translation models through lagrangian relaxation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 26–37, Stroudsburg, PA, USA. Association for Computational Linguistics.

Charniak, E. and Johnson, M. (2005). Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 173–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

Chatterjee, S. and Cancedda, N. (2010). Minimum error rate training by sampling the translation lattice. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 606–615, Stroudsburg, PA, USA. Association for Computational Linguistics.

Chiang, D. (2005). A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 263–270, Stroudsburg, PA, USA. Association for Computational Linguistics.

Chiang, D. (2007). Hierarchical phrase-based translation. *Computational Linguistics*, 33:201–228.

Chiang, D., Marton, Y., and Resnik, P. (2008). Online large-margin training of syntactic and structural translation features. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 224–233, Stroudsburg, PA, USA. Association for Computational Linguistics.

Cocke, J. and Schwartz, J. T. (1969). *Programming languages and their compilers: Preliminary notes*. Courant Institute of Mathematical Sciences, New York University.

Cohen, S. B., Simmons, R. J., and Smith, N. A. (2008). Dynamic programming algorithms as products of weighted logic programs. In Garcia de la Banda, M. and Pontelli, E., editors, *Logic Programming*, volume 5366 of *Lecture Notes in Computer Science*, pages 114–129. Springer Berlin Heidelberg.

Cormen, T. H., Stein, C., Rivest, R. L., and Leiserson, C. E. (2001). *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition.

Dantzig, G., Fulkerson, R., and Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Operations Research*, 2:393–410.

DeNero, J., Chiang, D., and Knight, K. (2009). Fast consensus decoding over translation forests. 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 2 - Volume 2*, ACL '09, pages 567–575, Stroudsburg, PA, USA. Association for Computational Linguistics.

Dyer, C. (2010). *A Formal Model of Ambiguity and its Applications in Machine Translation*. PhD thesis, University of Maryland.

Dyer, C., Muresan, S., and Resnik, P. (2008). Generalizing word lattice translation. In *Proceedings of ACL-08: HLT*, pages 1012–1020, Columbus, Ohio. Association for Computational Linguistics.

Dyer, C. and Resnik, P. (2010). Context-free reordering, finite-state translation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 858–866, Stroudsburg, PA, USA. Association for Computational Linguistics.

Dyer, C., Weese, J., Setiawan, H., Lopez, A., Ture, F., Eidelman, V., Ganitkevitch, J., Blunsom, P., and Resnik, P. (2010). cdec: a decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of the ACL 2010 System Demonstrations*, ACLDemos '10, pages 7–12, Stroudsburg, PA, USA. Association for Computational Linguistics.

Dymetman, M. (2013). Procedure for building a max-arpa table in order to compute optimistic back-offs in a language model.

Dymetman, M., Bouchard, G., and Carter, S. (2012a). Optimization and sampling for nlp from a unified viewpoint. In *Proceedings of the First International Workshop on Optimization Techniques for Human Language Technology*, pages 79–94, Mumbai, India. The COLING 2012 Organizing Committee.

Dymetman, M., Bouchard, G., and Carter, S. (2012b). The OS* Algorithm: a Joint Approach to Exact Optimization and Sampling. *ArXiv e-prints*.

Earley, J. (1970). An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102.

Eisner, J. and Tromble, R. W. (2006). Local search with very large-scale neighborhoods for optimal permutations in machine translation. In *Proceedings of the HLT-NAACL Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing.*, pages 57–75, New York.

Federico, M., Bertoldi, N., and Cettolo, M. (2008). Irstlm: an open source toolkit for handling large scale language models. In *9th Annual Conference of the International Speech Communication Association*, pages 1618–1621. ISCA.

Fredkin, E. (1960). Trie memory. *Commun. ACM*, 3(9):490–499.

Galley, M. and Manning, C. D. (2010). Accurate non-hierarchical phrase-based translation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 966–974, Stroudsburg, PA, USA. Association for Computational Linguistics.

Gallo, G., Longo, G., Pallottino, S., and Nguyen, S. (1993). Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2-3):177–201.

Gelman, A., Carlin, J., Stern, H., Dunson, D., Vehtari, A., and Rubin, D. (2013). *Bayesian Data Analysis.* Chapman and Hall/CRC, third edition.

Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6(6):721–741.

Germann, U., Jahr, M., Knight, K., Marcu, D., and Yamada, K. (2001). Fast decoding and optimal decoding for machine translation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, ACL '01, pages 228–235, Stroudsburg, PA, USA. Association for Computational Linguistics.

Germann, U., Joanis, E., and Larkin, S. (2009). Tightly packed tries: how to fit large models into memory, and make them load fast, too. In *Proceedings of the Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, SETQA-NLP '09, pages 31–39, Stroudsburg, PA, USA. Association for Computational Linguistics.

Gispert, A., Iglesias, G., Blackwood, G., R. Banga, E., and Byrne, W. (2010). Hierarchical phrase-based translation with weighted finite-state transducers and shallow-n grammars. *Comput. Linguist.*, 36(3):505–533.

Goodman, J. (1999). Semiring parsing. *Comput. Linguist.*, 25(4):573–605.

Goodman, J. T. (1998). *Parsing inside-out*. PhD thesis, Cambridge, MA, USA. AAI9832377.

Graehl, J. (2005). Relatively useless pruning. Technical report, USC Information Sciences Institute.

Greibach, S. A. (1965). A new normal-form theorem for context-free phrase structure grammars. *J. ACM*, 12(1):42–52.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107.

Heafield, K. (2011). Kenlm: faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, WMT '11, pages 187–197, Stroudsburg, PA, USA. Association for Computational Linguistics.

Heafield, K., Pouzyrevsky, I., Clark, J. H., and Koehn, P. (2013). Scalable modified kneser-ney language model estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 690–696, Sofia, Bulgaria. Association for Computational Linguistics.

Hopcroft, J. E. and Ullman, J. D. (1969). *Formal languages and their relation to automata*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction To Automata Theory, Languages, And Computation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

Hopkins, M. and Langmead, G. (2009). Cube pruning as heuristic search. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1*, EMNLP '09, pages 62–71, Stroudsburg, PA, USA. Association for Computational Linguistics.

Huang, L. (2008). *Forest-based algorithms in natural language processing*. PhD thesis, Philadelphia, PA, USA. AAI3346133.

Huang, L. and Chiang, D. (2007). Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 144–151, Prague, Czech Republic. Association for Computational Linguistics.

Iglesias, G., Allauzen, C., Byrne, W., de Gispert, A., and Riley, M. (2011). Hierarchical phrase-based translation representations. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1383, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition (Prentice Hall Series in Artificial Intelligence).* Prentice Hall, 1 edition.

Kasami, T. (1965). An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Knight, K. (1999). Decoding complexity in word-replacement translation models. *Comput. Linguist.*, 25(4):607–615.

Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. In *Tenth Machine Translation Summit*, pages 79 – 86.

Koehn, P., Hoang, H., Birch, A., Burch, C. C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. (2007). Moses: Open source toolkit for statistical machine translation. In *45th Annual Meeting of the Association for Computational Linguistics.*

Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 48–54, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kumar, S. and Byrne, W. (2003). A weighted finite state transducer implementation of the alignment template model for statistical machine translation. In *Proceedings of the 2003 Conference of the North American Chap-*

ter of the Association for Computational Linguistics on Human Language Technology - Volume 1, NAACL '03, pages 63–70, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kumar, S. and Byrne, W. (2004). Minimum bayes-risk decoding for statistical machine translation. In Susan Dumais, D. M. and Roukos, S., editors, *HLT-NAACL 2004: Main Proceedings*, pages 169–176, Boston, Massachusetts, USA. Association for Computational Linguistics.

Kumar, S., Deng, Y., and Byrne, W. (2006). A weighted finite state transducer translation template model for statistical machine translation. *Natural Language Engineering*, 12(1):35–75.

Kumar, S., Macherey, W., Dyer, C., and Och, F. (2009). Efficient minimum error rate training and minimum bayes-risk decoding for translation hypergraphs and lattices. 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*, pages 163–171, Suntec, Singapore. Association for Computational Linguistics.

Langlais, P., Patry, A., and Gotti, F. (2007). A greedy decoder for phrase-based statistical machine translation. In *11th International Conference on Theoretical and Methodological Issues in Machine Translation*, TMI, pages 104–113, SkZvde, Sweden,.

Lewis, II, P. M. and Stearns, R. E. (1968). Syntax-directed transduction. *Journal of the ACM*, 15(3):465–488.

Li, Z. and Eisner, J. (2009). First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1*, EMNLP '09, pages 40–51, Stroudsburg, PA, USA. Association for Computational Linguistics.

Li, Z., Eisner, J., and Khudanpur, S. (2009). Variational decoding for statistical machine translation. 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 2 - Volume 2*, ACL '09, pages 593–601, Stroudsburg, PA, USA. Association for Computational Linguistics.

Lopez, A. (2008). Statistical machine translation. *ACM Computing Surveys*, 40:8:1–8:49.

Lopez, A. (2009). Translation as weighted deduction. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '09, pages 532–540, Stroudsburg, PA, USA. Association for Computational Linguistics.

Marcu, D. and Wong, W. (2002). A phrase-based, joint probability model for statistical machine translation. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing - Volume 10*, EMNLP '02, pages 133–139, Stroudsburg, PA, USA. Association for Computational Linguistics.

May, J. and Knight, K. (2006). A better n-best list: Practical determinization of weighted finite tree automata. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, pages 351–358, Stroudsburg, PA, USA. Association for Computational Linguistics.

Melamed, I. D. (2003). Multitext grammars and synchronous parsers. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology -*

Volume 1, NAACL ’03, pages 79–86, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mohri, M. (2009). Weighted automata algorithms. In Droste, M., Kuich, W., and Vogler, H., editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science. An EATCS Series, pages 213–254. Springer Berlin Heidelberg.

Mohri, M., Pereira, F. C. N., and Riley, M. (1996). Weighted automata in text and speech processing. In *Proceedings of the 12th biennial European Conference on Artificial Intelligence (ECAI-96), Workshop on Extended finite state models of language*. John Wiley and Sons.

Nederhof, M.-J. and Satta, G. (2008a). Computing partition functions of pcfgs. *Research on Language and Computation*, 6(2):139–162.

Nederhof, M.-J. and Satta, G. (2008b). Probabilistic parsing. In Bel-Enguix, G., Jiménez-López, M. D., and Martín-Vide, C., editors, *New Developments in Formal Languages and Applications, Studies in Computational Intelligence*, volume 113, pages 229–258. Springer.

Och, F. J. (2003). Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, volume 1 of *ACL ’03*, pages 160–167, Stroudsburg, PA, USA. Association for Computational Linguistics.

Och, F. J. and Ney, H. (2002). Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL ’02, pages 295–302, Stroudsburg, PA, USA. Association for Computational Linguistics.

Och, F. J. and Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29:19–51.

Och, F. J., Ueffing, N., and Ney, H. (2001). An efficient a* search algorithm for statistical machine translation. In *Proceedings of the workshop on Data-driven methods in machine translation - Volume 14*, DMMT '01, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 311–318.

Pereira, F. C. N. and Warren, D. H. D. (1983). Parsing as deduction. In *Proceedings of the 21st annual meeting on Association for Computational Linguistics*, ACL '83, pages 137–144, Stroudsburg, PA, USA. Association for Computational Linguistics.

Riedel, S. and Clarke, J. (2009). Revisiting optimal decoding for machine translation ibm model 4. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, NAACL-Short '09, pages 5–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Robert, C. P. and Casella, G. (2004). *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Roth, B., McCallum, A., Dymetman, M., and Cancedda, N. (2010). Machine translation using overlapping alignments and sample rank. In *The Ninth Conference of the Association for Machine Translation in the Americas*, Dever, Colorado.

Rudin, W. (1987). *Real and Complex Analysis*. McGraw-Hill.

Rush, A., Chang, Y.-W., and Collins, M. (2013). Optimal beam search for machine translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 210–221, Seattle, Washington, USA. Association for Computational Linguistics.

Rush, A. M. and Collins, M. (2011). Exact decoding of syntactic translation models through lagrangian relaxation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 72–82, Stroudsburg, PA, USA. Association for Computational Linguistics.

Rush, A. M., Sontag, D., Collins, M., and Jaakkola, T. (2010). On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Cambridge, MA. Association for Computational Linguistics.

Schabes, Y. (1990). *Mathematical and Computational Aspects of Lexicalized Grammars*. PhD thesis, Philadelphia, PA, USA. AAI9101213.

Shieber, S. M. and Schabes, Y. (1990). Synchronous tree-adjoining grammars. In *Proceedings of the 13th Conference on Computational Linguistics - Volume 3*, COLING '90, pages 253–258, Stroudsburg, PA, USA. Association for Computational Linguistics.

Shieber, S. M., Schabes, Y., and Pereira, F. C. N. (1995). Principles and implementation of deductive parsing. *Journal of Logic Programming*, pages 24:3–36.

Sima'an, K. (1996). Computational complexity of probabilistic disambiguation by means of tree-grammars. In *Proceedings of the 16th Conference on*

Computational Linguistics, volume 2 of *COLING '96*, pages 1175–1180, Stroudsburg, PA, USA. Association for Computational Linguistics.

Sima'an, K. (2002). Computational complexity of probabilistic disambiguation. *Grammars*, 5(2):125–151.

Simard, M., Cancedda, N., Cavestro, B., Dymetman, M., Gaussier, E., Goutte, C., Yamada, K., Langlais, P., and Mauser, A. (2005). Translating with non-contiguous phrases. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 755–762, Stroudsburg, PA, USA. Association for Computational Linguistics.

Sixtus, A. and Ortmanns, S. (1999). High quality word graphs using forward-backward pruning. In *Proceedings of the Acoustics, Speech, and Signal Processing, 1999. on 1999 IEEE International Conference - Volume 02*, ICASSP '99, pages 593–596, Washington, DC, USA. IEEE Computer Society.

Stolcke, A. (2002). Srilm - an extensible language modeling toolkit. In *Proceedings of the International Conference of Spoken Language Processing (INTERSPEECH 2002)*, pages 257–286.

Tillmann, C. and Ney, H. (2003). Word reordering and a dynamic programming beam search algorithm for statistical machine translation. *Comput. Linguist.*, 29(1):97–133.

Tillmann, C., Vogel, S., Ney, H., and Zubiaga, A. (1997). A dp based search using monotone alignments in statistical translation. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, EACL '97, pages 289–296, Stroudsburg, PA, USA. Association for Computational Linguistics.

Tromble, R. W., Kumar, S., Och, F., and Macherey, W. (2008). Lattice minimum bayes-risk decoding for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 620–629, Stroudsburg, PA, USA. Association for Computational Linguistics.

Wang, Y.-Y. and Waibel, A. (1997). Decoding algorithm in statistical machine translation. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, EACL '97, pages 366–372, Stroudsburg, PA, USA. Association for Computational Linguistics.

Watanabe, T., Tsukada, H., and Isozaki, H. (2006). Left-to-right target generation for hierarchical phrase-based translation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, pages 777–784, Stroudsburg, PA, USA. Association for Computational Linguistics.

Wick, M., Rohanimanesh, K., Culotta, A., and Mccallum, A. (2009). Samplerank: Learning preference from atomic gradients. In *Neural Information Processing Systems (NIPS), Workshop on Advances in Ranking.*

Wu, D. (1995). Stochastic inversion transduction grammars with application to segmentation, bracketing, and alignment of parallel corpora. In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2*, IJCAI'95, pages 1328–1335, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Wu, D. (1997). Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–403.

Younger, D. H. (1966). Context-free language processing in time n3. In *Proceedings of the 7th Annual Symposium on Switching and Automata Theory (swat 1966)*, SWAT '66, pages 7–20, Washington, DC, USA. IEEE Computer Society.

Zaslavskiy, M., Dymetman, M., and Cancedda, N. (2009). Phrase-based statistical machine translation as a traveling salesman problem. 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 333–341, Stroudsburg, PA, USA. Association for Computational Linguistics.

Appendix A

Publications

The following is a list of all publications produced during this PhD.

[1] Aziz, W., Koponen, M., and Specia, L., Sub-sentence level analysis of machine translation post-editing effort, in *Post-editing of Machine Translation: Processes and Applications*, edited by O'Brien, S., Balling, L. W., Carl, M., Simard, M., and Specia, L., chapter 8, Cambridge Scholars Publishing, 2014.

[2] Aziz, W., Dymetman, M., and Venkatapathy, S., Investigations in exact inference for hierarchical translation, in *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 472–483, Sofia, Bulgaria, 2013, Association for Computational Linguistics.

[3] Aziz, W. and Specia, L., Multilingual wsd-like constraints for paraphrase extraction, in *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 202–211, Sofia, Bulgaria, 2013, Association for Computational Linguistics.

[4] Aziz, W., Mitkov, R., and Specia, L., Ranking machine translation systems via post-editing, in *In Proceedings of Text, Speech and Dialogue (TSD)*, volume 8082 of *Lecture Notes in Computer Science*, pages 410–418, Pilsen, Czech Republic, 2013, Springer Berlin Heidelberg.

[5] Maarit Koponen, Wilker Aziz, L. R. and Specia, L., Post-editing time as a measure of cognitive effort , in *AMTA 2012 Workshop on Post-Editing Technology and Practice (WPTP 2012)*, pages 11–20, San Diego, USA, 2012, Association for Machine Translation in the Americas (AMTA).

[6] Rios, M., Aziz, W., and Specia, L., UOW: Semantically informed text similarity, in **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 673–678, Montréal, Canada, 2012, Association for Computational Linguistics.

[7] Aziz, W., de Sousa, S. C. M., and Specia, L., PET: a tool for post-editing and assessing machine translation, in *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, 2012, European Language Resources Association (ELRA).

[8] Aziz, W. and Specia, L., PET: a tool for post-editing and assessing machine translation, in *The 16th Annual Conference of the European Association for Machine Translation*, EAMT '12, page 99, Trento, Italy, 2012.

[9] Aziz, W., de Sousa, S. C. M., and Specia, L., Cross-lingual sentence compression for subtitles, in *The 16th Annual Conference of the European Association for Machine Translation*, EAMT '12, pages 103–110, Trento, Italy, 2012.

[10] Specia, L., Hajlaoui, N., Hallett, C., and Aziz, W., Predicting machine translation adequacy, in *Proceedings of the 13th Machine Translation Summit*, pages 513–520, Xiamen, China, 2011.

[11] Aziz, W., Rios, M., and Specia, L., Shallow semantic trees for smt, in *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 316–322, Edinburgh, Scotland, 2011, Association for Computational Linguistics.

[12] Rios, M., Aziz, W., and Specia, L., Tine: A metric to assess mt adequacy, in *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 116–122, Edinburgh, Scotland, 2011, Association for Computational Linguistics.

[13] Aziz, W., Rios, M., and Specia, L., Improving chunk-based semantic role labeling with lexical features, in *Proceedings of the International Conference Recent Advances in Natural Language Processing 2011*, pages 226–232, Hissar, Bulgaria, 2011, RANLP 2011 Organising Committee.

[14] de Sousa, S. C. M., Aziz, W., and Specia, L., Assessing the post-editing effort for automatic and semi-automatic translations of DVD subtitles, in *Proceedings of the International Conference Recent Advances in Natural Language Processing 2011*, pages 97–103, Hissar, Bulgaria, 2011, RANLP 2011 Organising Committee.

[15] Aziz, W. and Specia, L., Fully automatic compilation of a Portuguese-English parallel corpus for statistical machine translation, in *Proceedings of the 8th Brazilian Symposium in Information and Human Language Technology*, Cuiabá, MT, 2011.