SCFG Decoding Without Binarization

Mark Hopkins and Greg Langmead
SDL Language Weaver, Inc.
6060 Center Drive, Suite 150
Los Angeles, CA 90045
{mhopkins,glangmead}@languageweaver.com

Abstract

Conventional wisdom dictates that synchronous context-free grammars (SCFGs) must be converted to Chomsky Normal Form (CNF) to ensure cubic time decoding. For arbitrary SCFGs, this is typically accomplished via the synchronous binarization technique of (Zhang et al., 2006). A drawback to this approach is that it inflates the constant factors associated with decoding, and thus the practical running time. (DeNero et al., 2009) tackle this problem by defining a superset of CNF called Lexical Normal Form (LNF), which also supports cubic time decoding under certain implicit assumptions. In this paper, we make these assumptions explicit, and in doing so, show that LNF can be further expanded to a broader class of grammars (called "scope-3") that also supports cubic-time decoding. By simply pruning non-scope-3 rules from a GHKM-extracted grammar, we obtain better translation performance than synchronous binarization.

1 Introduction

At the heart of bottom-up chart parsing (Younger, 1967) is the following combinatorial problem. We have a context-free grammar (CFG) rule (for instance, S → NP VP PP) and an input sentence of length n (for instance, "on the fast jet ski of mr smith"). During chart parsing, we need to apply the rule to all relevant subspans of the input sentence. See Figure 1. For this particular rule, there are $\binom{n+1}{4}$ *application contexts*, i.e. ways to choose the subspans. Since the asymptotic running time of chart parsing is at least linear in this quantity, it will take at least $O(\binom{n+1}{4}) = O(n^4)$ time if we include this rule in our grammar.

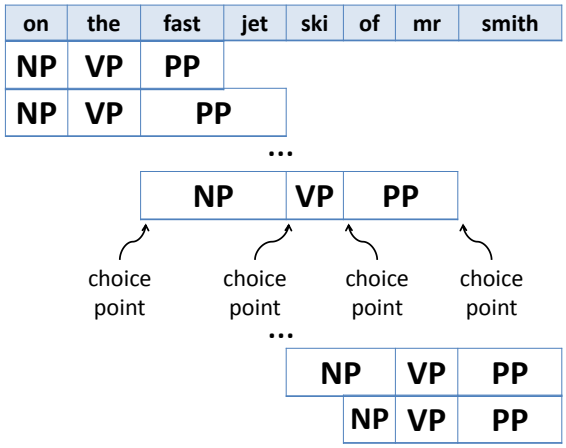

Figure 1: A demonstration of application contexts. There are $\binom{n+1}{4}$ application contexts for the CFG rule "S → NP VP PP", where n is the length of the input sentence.

Fortunately, we can take advantage of the fact that any CFG has an equivalent representation in Chomsky Normal Form (CNF). In CNF, all rules have the form X → Y Z or X → x, where x is a terminal and X, Y, Z are nonterminals. If a rule has the form X → Y Z, then there are only $\binom{n+1}{3}$ application contexts, thus the running time of chart parsing is $O(\binom{n+1}{3}) = O(n^3)$ when applied to CNF grammars.

A disadvantage to CNF conversion is that it increases both the overall number of rules and the overall number of nonterminals. This inflation of the "grammar constant" does not affect the asymptotic runtime, but can have a significant impact on the performance in practice. For this reason, (DeN-

Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, pages 646–655,
MIT, Massachusetts, USA, 9-11 October 2010.

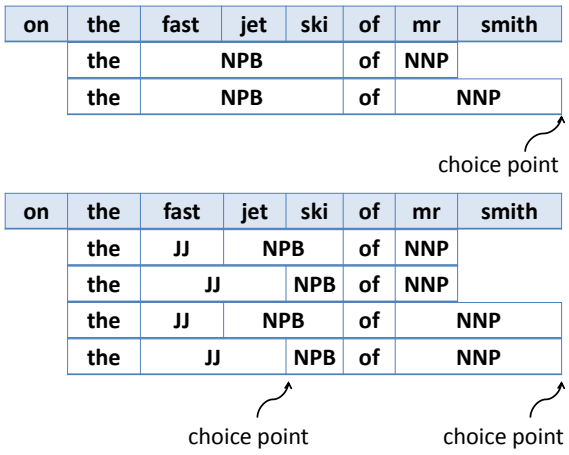

Figure 2: A demonstration of application contexts for rules with lexical anchors. There are $O(n)$ application contexts for CFG rule "S $\rightarrow$ the NPB of NNP", and $O(n^2)$ application contexts for CFG rule "S $\rightarrow$ the JJ NPB of NNP", if we assume that the input sentence has length n and contains no repeated words.

ero et al., 2009) provide a relaxation of CNF called Lexical Normal Form (LNF). LNF is a superclass of CNF that also allows rules whose right-hand sides have no consecutive nonterminals. The intuition is that the terminals provide anchors that limit the applicability of a given rule. For instance, consider the rule NP $\rightarrow$ the NPB of NNP. See Figure 2. Because the terminals constrain our choices, there are only two different application contexts. The implicit assumption is that input sentences will not repeat the same word more than a small constant number of times. If we make the explicit assumption that all words of an input sentence are unique, then there are $O(n^2)$ application contexts for a "no consecutive nonterminals" rule. Thus under this assumption, the running time of chart parsing is still $O(n^3)$ when applied to LNF grammars.

But once we make this assumption explicit, it becomes clear that we can go even further than LNF and still maintain the cubic bound on the runtime. Consider the rule NP $\rightarrow$ the JJ NPB of NNP. This rule is not LNF, but there are still only $O(n^2)$ application contexts, due to the anchoring effect of the terminals. In general, for a rule of the form X $\rightarrow \gamma$, there are at most $O(n^p)$ application contexts, where p is the number of consecutive nonterminal pairs in the string X $\cdot\gamma\cdot$ X (where X is an arbitrary nonterminal). We refer to p as the *scope* of a rule. Thus chart parsing runs in time $O(n^{\text{scope}(G)})$, where $\text{scope}(G)$ is the maximum scope of any of the rules in CFG G. Specifically, any scope-3 grammar can be decoded in cubic time.

Like (DeNero et al., 2009), the target of our interest is synchronous context-free grammar (SCFG) decoding with rules extracted using the GHKM algorithm (Galley et al., 2004). In practice, it turns out that only a small percentage of the lexical rules in our system have scope greater than 3. By simply removing these rules from the grammar, we can maintain the cubic running time of chart parsing without any kind of binarization. This has three advantages. First, we do not inflate the grammar constant. Second, unlike (DeNero et al., 2009), we maintain the *synchronous* property of the grammar, and thus can integrate language model scoring into chart parsing. Finally, a system without binarized rules is considerably simpler to build and maintain. We show that this approach gives us better practical performance than a mature system that binarizes using the technique of (Zhang et al., 2006).

2 Preliminaries

Assume we have a global vocabulary of *symbols*, containing the reserved *substitution symbol* $\Diamond$. Define a *sentence* as a sequence of symbols. We will typically use space-delimited quotations to represent example sentences, e.g. "the fast jet ski" rather than $\langle$the, fast, jet, ski$\rangle$. We will use the dot operator to represent the concatenation of sentences, e.g. "the fast" $\cdot$ "jet ski" = "the fast jet ski".

Define the *rank* of a sentence as the count of its $\Diamond$ symbols. We will use the notation $\text{SUB}(s, s_1, ..., s_k)$ to denote the substitution of k sentences $s_1, ..., s_k$ into a k-rank sentence s. For instance, if s = "the $\Diamond$ $\Diamond$ of $\Diamond$", then $\text{SUB}(s,$ "fast", "jet ski", "mr smith"$)$ = "the fast jet ski of mr smith".

To refer to a subsentence, define a *span* as a pair $[a, b]$ of nonnegative integers such that $a < b$. For a sentence $s = \langle s_1, s_2, ..., s_n\rangle$ and a span $[a, b]$ such that $b \leq n$, define $s_{[a,b]} = \langle s_{a+1}, ..., s_b\rangle$.

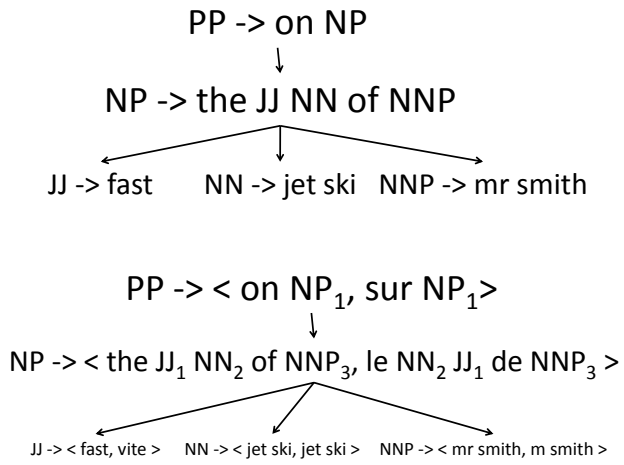

Figure 3: An example CFG derivation (above) and an example SCFG derivation (below). Both derive the sentence SUB("on ◊", SUB("the ◊ ◊ of ◊", "fast", "jet ski", "mr smith")) = "on the fast jet ski of mr smith". The SCFG derivation simultaneously derives the auxiliary sentence "sur le jet ski vite de m smith".

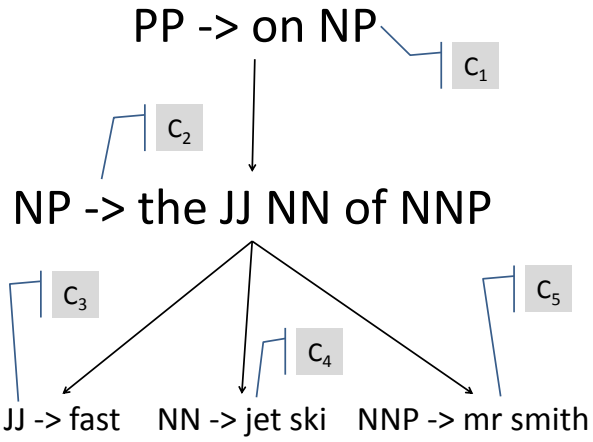

Figure 4: The cost of the CFG derivation in Figure 3 is $C_1 + C_2 + C_3 + C_4 + C_5$, where C_1 is the base cost of rule "PP → on NP", etc. Notice that this cost can be distributed locally to the nodes of the derivation.

3 Minimum Derivation Cost

Chart parsing solves a problem which we will refer to as Minimum Derivation Cost. Because we want our results to be applicable to both CFG decoding and SCFG decoding with an integrated language model, we will provide a somewhat more abstract formulation of chart parsing than usual.

In Figure 3, we show an example of a CFG derivation. A derivation is a tree of CFG rules, constructed so that the *preconditions* (the RHS nonterminals) of any rule match the *postconditions* (the LHS nonterminal) of its child rules. The purpose of a derivation is to *derive* a sentence, which is obtained through recursive substitution. In the example, we substitute "fast", "jet ski", and "mr smith" into the lexical *pattern* "the ◊ ◊ of ◊" to obtain "the fast jet ski of mr smith". Then we substitute this result into the lexical pattern "on ◊" to obtain "on the fast jet ski of mr smith".

The cost of a derivation is simply the sum of the base costs of its rules. Thus the cost of the CFG derivation in Figure 3 is $C_1 + C_2 + C_3 + C_4 + C_5$, where C_1 is the base cost of rule "PP → on NP", etc. Notice that this cost can be distributed locally to the nodes of the derivation (Figure 4).

An SCFG derivation is similar to a CFG derivation, except that it simultaneously derives two sentences. For instance, the SCFG derivation in Figure 3 derives the sentence pair ⟨ "on the fast jet ski of mr smith", "sur le jet ski vite de m smith" ⟩. In machine translation, often we want the cost of the SCFG derivation to include a language model cost for this second sentence. For example, the cost of the SCFG derivation in Figure 3 might be $C_1+C_2+C_3+C_4+C_5+LM(\text{sur le})+LM(\text{le jet})+LM(\text{jet ski})+LM(\text{ski de})+LM(\text{de m})+LM(\text{m smith})$, where LM is the negative log of a 2-gram language model. This new cost function can also be distributed locally to the nodes of the derivation, as shown in Figure 5. However, in order to perform the local computations, we need to pass information (in this case, the LM boundary words) up the tree. We refer to this extra information as *carries*. Formally, define a *carry* as a sentence of rank 0.

In order to provide a chart parsing formulation that applies to both CFG decoding and SCFG decoding with an integrated language model, we need abstract definitions of rule and derivation that capture the above concepts of *pattern*, *postcondition*, *preconditions*, *cost*, and *carries*.

3.1 Rules

Define a *rule* as a tuple $\langle k, s^*, X, \pi, \Gamma, c \rangle$, where k is a nonnegative integer called the *rank*, s^* is a rank-k

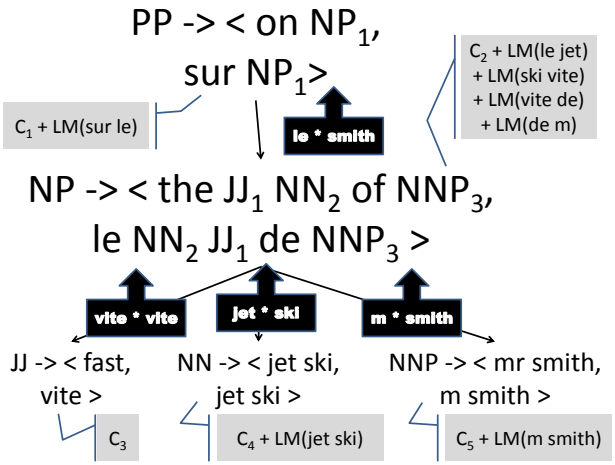

Figure 5: The cost of the SCFG derivation in Figure 3 (with an integrated language model score) can also be distributed to the nodes of the derivation, but to perform the local computations, information must be passed up the tree. We refer to this extra information as a *carry*.

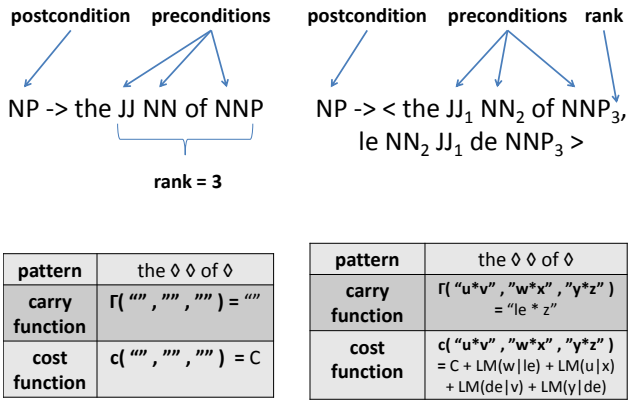

Figure 6: Deconstruction of a CFG rule (left) and SCFG rule (right) according to the definition of rule in Section 3.1. The carry function of the SCFG rule computes boundary words for a 2-gram language model. In the cost functions, C is a real number and LM returns the negative log of a language model query.

sentence called the *pattern* [1], X is a symbol called the *postcondition*, π is a k-length sentence called the *preconditions*, Γ is a function (called the *carry function*) that maps a k-length list of carries to a carry, and c is a function (called the *cost function*) that maps a k-length list of carries to a real number. Figure 6 shows a CFG and an SCFG rule, deconstructed according to this definition. [2] Note that the CFG rule has trivial cost and carry functions that map everything to a constant. We refer to such rules as *simple*.

We will use $\mathsf{post}(r)$ to refer to the postcondition of rule r, and $\mathsf{pre}(r, i)$ to refer to the i^{th} precondition of rule r.

Finally, define a *grammar* as a finite set of rules. A grammar is *simple* if all its rules are simple.

3.2 Derivations

For a grammar R, define $\mathsf{deriv}(R)$ as the smallest set that contains every tuple $\langle r, \delta_1, ..., \delta_k \rangle$ satisfying the following conditions:

- $r \in R$ is a k-rank rule
- $\delta_i \in \mathsf{deriv}(R)$ for all $1 \leq i \leq k$
- $\mathsf{pre}(r, i) = \mathsf{post}(r_i)$ for all $1 \leq i \leq k$, where r_i is the first element of tuple δ_i.

An *R–derivation* is an element of $\mathsf{deriv}(R)$. Consider a derivation $\delta = \langle r, \delta_1, ..., \delta_k \rangle$, where rule $r = \langle k, s^*, X, \pi, \Gamma, c \rangle$. Define the following properties:

$$\mathsf{post}(\delta) = \mathsf{post}(r)$$
$$\mathsf{sent}(\delta) = \textsc{sub}(s^*, \mathsf{sent}(\delta_1), ..., \mathsf{sent}(\delta_k))$$
$$\mathsf{carry}(\delta) = \Gamma(\mathsf{carry}(\delta_1), ..., \mathsf{carry}(\delta_k))$$
$$\mathsf{cost}(\delta) = c(\mathsf{carry}(\delta_1), ..., \mathsf{carry}(\delta_k)) + \sum_{j=1}^{k} \mathsf{cost}(\delta_j)$$

In words, we say that derivation δ *derives* sentence $\mathsf{sent}(\delta)$. If for some span σ of a particular sentence s, it holds that $\mathsf{sent}(\delta) = s_\sigma$, then we will say that δ is a derivation *over* span σ.

3.3 Problem Statement

The Minimum Derivation Cost problem is the following. Given a set R of rules and an input sentence

[1] For simplicity, we also impose the condition that "◊" is not a valid pattern. This is tantamount to disallowing unary rules.

[2] One possible point of confusion is why the pattern of the SCFG rule refers only to the primary sentence, and not the auxiliary sentence. To reconstruct the auxiliary sentence from an SCFG derivation in practice, one would need to augment the abstract definition of rule with an auxiliary pattern. However this is not required for our theoretical results.

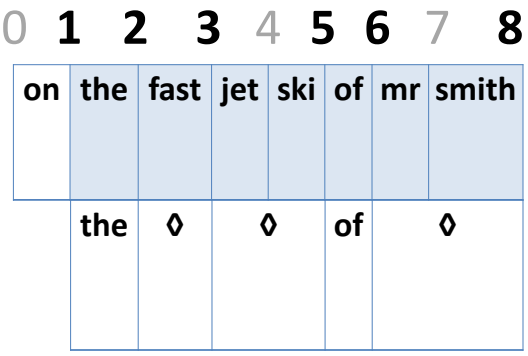

Figure 7: An application context for the pattern "the ◊ ◊ of ◊" and the sentence "on the fast jet ski of mr smith".

s, find the minimum cost of any R–derivation that derives s. In other words, compute:

$$\mathsf{MinDCost}(R, s) \triangleq \min_{\delta \in \mathsf{deriv}(R) | \mathsf{sent}(\delta) = s} \mathsf{cost}(\delta)$$

4 Application Contexts

Chart parsing solves Minimum Derivation Cost via dynamic programming. It works by building derivations over increasingly larger spans of the input sentence s. Consider just one of these spans σ. How do we build a derivation over that span?

Recall that a derivation takes the form $\langle r, \delta_1, ..., \delta_k \rangle$. Given the rule r and its pattern s^*, we need to choose the subderivations δ_i such that $\textsc{sub}(s^*, \mathsf{sent}(\delta_1), ..., \mathsf{sent}(\delta_k)) = s_\sigma$. To do so, we must match the pattern to the span, so that we know which subspans we need to build the subderivations over. Figure 7 shows a matching of the pattern "the ◊ ◊ of ◊" to span $[1, 8]$ of the sentence "on the fast jet ski of mr smith". It tells us that we can build a derivation over span $[1, 8]$ by choosing this rule and subderivations over subspans $[2, 3]$, $[3, 5]$, and $[6, 8]$.

We refer to these matchings as *application contexts*. Formally, given two sentences s^* and s of respective lengths m and n, define an $\langle s^*, s \rangle$–*context* as an monotonically increasing sequence $\langle x_0, x_1, ..., x_m \rangle$ of integers between 0 and n such that for all i:

$$s^*_{[i-1,i]} \neq \Diamond \text{ implies that } s^*_{[i-1,i]} = s_{[x_{i-1}, x_i]}$$

The context shown in Figure 7 is $\langle 1, 2, 3, 5, 6, 8 \rangle$. Use $\mathsf{cxt}(s^*, s)$ to denote the set of all $\langle s^*, s \rangle$–contexts.

An $\langle s^*, s \rangle$–context $x = \langle x_0, x_1, ..., x_m \rangle$ has the following properties:

$$\mathsf{span}(x; s^*, s) = [x_0, x_m]$$
$$\mathsf{subspans}(x; s^*, s) = \langle [x_0, x_1], ..., [x_{m-1}, x_m] \rangle$$

Moreover, define $\mathsf{varspans}(x; s^*, s)$ as the subsequence of $\mathsf{subspans}(x; s^*, s)$ including only $[x_{i-1}, x_i]$ such that $s^*_{[i-1,i]} = \Diamond$. For the context x shown in Figure 7:

$$\mathsf{span}(x; s^*, s) = [1, 8]$$
$$\mathsf{subspans}(x; s^*, s) = \langle [1, 2], [2, 3], [3, 5], [5, 6], [6, 8] \rangle$$
$$\mathsf{varspans}(x; s^*, s) = \langle [2, 3], [3, 5], [6, 8] \rangle$$

An application context $x \in \mathsf{cxt}(s^*, s)$ tells us that we can build a derivation over $\mathsf{span}(x)$ by choosing a rule with pattern s^* and subderivations over each span in $\mathsf{varspans}(x; s^*, s)$.

5 Chart Parsing Algorithm

We are now ready to describe the chart parsing algorithm. Consider a span σ of our input sentence s and assume that we have computed and stored all derivations over any subspan of σ. A naive way to compute the minimum cost derivation over span σ is to consider every possible derivation:

1. Choose a rule $r = \langle k, s^*, X, \pi, \Gamma, c \rangle$.
2. Choose an application context $x \in \mathsf{cxt}(s^*, s)$ such that $\mathsf{span}(x; s^*, s) = \sigma$.
3. For each subspan $\sigma_i \in \mathsf{varspans}(x; s^*, s)$, choose a subderivation δ_i such that $\mathsf{post}(\delta_i) = \mathsf{pre}(r, i)$.

The key observation here is the following. In order to score such a derivation, we did not actually need to know each subderivation in its entirety. We merely needed to know the following information about it: (a) the subspan that it derives, (b) its postcondition, (c) its carry.

Chart parsing takes advantage of the above observation to avoid building all possible derivations. Instead it groups together derivations that share a common subspan, postcondition, and carry, and records only the minimum cost for each equivalence class. It records this cost in an associative map referred to as the *chart*.

Specifically, assume that we have computed and stored the minimum cost of every derivation class $\langle \sigma', X', \gamma' \rangle$, where X' is a postcondition, γ' is a carry, and σ' is a proper subspan of σ. Chart parsing computes the minimum cost of every derivation class $\langle \sigma, X, \gamma \rangle$ by adapting the above naive method as follows:

1. Choose a rule $r = \langle k, s^*, X, \pi, \Gamma, c \rangle$.
2. Choose an application context $x \in \mathsf{cxt}(s^*, s)$ such that $\mathsf{span}(x; s^*, s) = \sigma$.
3. For each subspan $\sigma_i \in \mathsf{varspans}(x; s^*, s)$, **choose a derivation class** $\langle \sigma_i, X_i, \gamma_i \rangle$ **from the chart** such that $X_i = \mathsf{pre}(r, i)$.
4. **Update**[3] **the cost of derivation class** $\langle \sigma, \mathsf{post}(r), \Gamma(\gamma_1, ..., \gamma_k) \rangle$ **with:**

$$c(\gamma_1, ..., \gamma_k) + \sum_{i=1}^{k} \mathsf{chart}[\sigma_i, X_i, \gamma_i]$$

where $\mathsf{chart}[\sigma_i, X_i, \gamma_i]$ **refers to the stored cost of derivation class** $\langle \sigma_i, X_i, \gamma_i \rangle$.

By iteratively applying the above method to all subspans of size 1, 2, etc., chart parsing provides an efficient solution for the Minimum Derivation Cost problem.

6 Runtime Analysis

At the heart of chart parsing is a single operation: the updating of a value in the chart. The running time is linear in the number of these chart updates. [4] The typical analysis counts the number of chart updates *per span*. Here we provide an alternative analysis that counts the number of chart updates *per rule*. This provides us with a finer bound with practical implications.

Let r be a rule with rank k and pattern s^*. Consider the chart updates involving rule r. There is (potentially) an update for every choice of (a) span, (b) application context, and (c) list of k derivation classes. If we let $\mathfrak{C}$ be the set of possible carries, then this means there are at most $|\mathsf{cxt}(s^*, s)| \cdot |\mathfrak{C}|^k$ updates involving rule r. [5] If we are doing beam decoding (i.e. after processing a span, the chart keeps only the B items of lowest cost), then there are at most $|\mathsf{cxt}(s^*, s)| \cdot B^k$ updates.

We can simplify the above by providing an upper bound for $|\mathsf{cxt}(s^*, s)|$. Define an *ambiguity* as the sentence "$\Diamond$ $\Diamond$", and define $\mathsf{scope}(s^*)$ as the number of ambiguities in the sentence "$\Diamond$" $\cdot s^* \cdot$ "$\Diamond$". The following bound holds:

Lemma 1. *Assume that a zero-rank sentence s does not contain the same symbol more than once. Then* $|\mathsf{cxt}(s^*, s)| \leq |s|^{\mathsf{scope}(s^*)}$.

Proof. Suppose s^* and s have respective lengths m and n. Consider $\langle x_0, x_1, ..., x_m \rangle \in \mathsf{cxt}(s^*, s)$. Let I be the set of integers i between 1 and m such that $s_i^* \neq \Diamond$ and let I^+ be the set of integers i between 0 and $m-1$ such that $s_{i+1}^* \neq \Diamond$. If $i \in I$, then we know the value of x_i, namely it is the unique integer j such that $s_j = s_i^*$. Similarly, if $i \in I^+$, then the value of x_i must be the unique integer j such that $s_j = s_{i+1}^*$. Thus the only nondetermined elements of context x_i are those for which $i \notin I \cup I^+$. Hence $|\mathsf{cxt}(s^*, s)| \leq |s|^{\{0,1,...,m\} - I - I^+} = |s|^{\mathsf{scope}(s^*)}$. □

Hence, under the assumption that the input sentence s does not contain the same symbol more than once, then there are at most $|s|^{\mathsf{scope}(s^*)} \cdot |\mathfrak{C}|^k$ chart updates involving a rule with pattern s^*.

For a rule r with pattern s^*, define $\mathsf{scope}(r) = \mathsf{scope}(s^*)$. For a grammar R, define $\mathsf{scope}(R) = \max_{r \in R} \mathsf{scope}(r)$ and $\mathsf{rank}(R) = \max_{r \in R} \mathsf{rank}(r)$.

Given a grammar R and an input sentence s, the above lemma tells us that chart parsing makes

[3] Here, *update* means "replace the cost associated with the class if the new cost is lower."

[4] This assumes that you can linearly enumerate the relevant updates. One convenient way to do this is to frame the enumeration problem as a search space, e.g. (Hopkins and Langmead, 2009)

[5] For instance, in SCFG decoding with an integrated j-gram language model, a carry consists of $2(j-1)$ boundary words. Generally it is assumed that there are $O(n)$ possible choices for a boundary word, and hence $O(n^{2(j-1)})$ possible carries.

$O(|s|^{\mathsf{scope}(R)} \cdot |\mathfrak{C}|^{\mathsf{rank}(R)})$ chart updates. If we restrict ourselves to beam search, than chart parsing makes $O(|s|^{\mathsf{scope}(R)})$ chart updates. [6]

6.1 On the Uniqueness Assumption

In practice, it will not be true that each input sentence contains only unique symbols, but it is not too far removed from the practical reality of many use cases, for which relatively few symbols repeat themselves in a given sentence. The above lemma can also be relaxed to assume only that there is a constant upper bound on the multiplicity of a symbol in the input sentence. This does not affect the O-bound on the number of chart updates, as long as we further assume a constant limit on the length of rule patterns.

7 Scope Reduction

From this point of view, CNF binarization can be viewed as a specific example of *scope reduction*. Suppose we have a grammar R of scope p. See Figure 8. If we can find a grammar $\hat{R}$ of scope $\hat{p} < p$ which is "similar" to grammar R, then we can decode in $O(n^{\hat{p}})$ rather than $O(n^p)$ time.

We can frame the problem by assuming the following parameters:

- a grammar R
- a desired scope p
- a loss function Λ that returns a (non-negative real-valued) score for any two grammars R and $\hat{R}$; if $\Lambda(R, \hat{R}) = 0$, then the grammars are considered to be equivalent

A *scope reduction method with loss* λ finds a grammar $\hat{R}$ such that $\mathsf{scope}(\hat{R}) \leq p$ and $\Lambda(R, \hat{R}) = \lambda$. A scope reduction method is *lossless* when its loss is 0.

In the following sections, we will use the loss function:

$$\Lambda(R, \hat{R}) = |\mathsf{MinDCost}(R, s) - \mathsf{MinDCost}(\hat{R}, s)|$$

where s is a fixed input sentence. Observe that if $\Lambda(R, \hat{R}) = 0$, then the solution to the Minimum Derivation Cost problem is the same for both R and $\hat{R}$. [7]

[6] Assuming $\mathsf{rank}(R)$ is bounded by a constant.

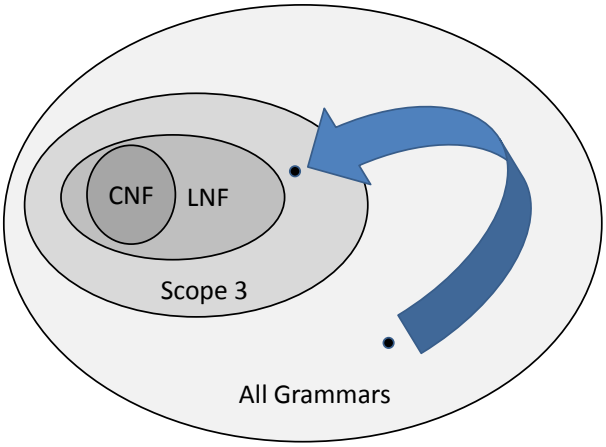

Figure 8: The "scope reduction" problem. Given a grammar of large scope, find a similar grammar of reduced scope.

7.1 CNF Binarization

A rule r is *CNF* if its pattern is "◊ ◊" or "x", where x is any non-substitution symbol. A grammar is *CNF* if all of its rules are CNF. Note that the maximum scope of a CNF grammar is 3.

CNF binarization is a deterministic process that maps a simple grammar to a CNF grammar. Since binarization takes subcubic time, we can decode with any grammar R in $O(n^3)$ time by converting R to CNF grammar $\hat{R}$, and then decoding with $\hat{R}$. This is a lossless scope reduction method.

What if grammar R is not simple? For SCFG grammars, (Zhang et al., 2006) provide a scope reduction method called *synchronous binarization* with quantifiable loss. Synchronous binarization selects a "binarizable" subgrammar R' of grammar R, and then converts R' into a CNF grammar $\hat{R}$. The cost and carry functions of these new rules are constructed such that the conversion from R' to $\hat{R}$ is a lossless scope reduction. Thus the total loss of the method is $|\mathsf{MinDCost}(R, s) - \mathsf{MinDCost}(R', s)|$. Fortunately, they find in practice that R' usually contains the great majority of the rules of R, thus they

[7] Note that if we want the actual derivation and not just its cost, then we need to specify a more finely grained loss function. This is omitted for clarity and left as an exercise.

a ◊ ◊	a b ◊ ◊ c
◊ ◊ a	◊ a ◊ ◊ b
a ◊ ◊ b	a ◊ b ◊ ◊
◊ a ◊ ◊	◊ ◊ a ◊ b
◊ ◊ a ◊	◊ ◊ ◊ a b
◊ ◊ ◊ a	a ◊ ◊ ◊ b
a ◊ ◊ ◊	a ◊ ◊ b ◊ c
a b ◊ ◊	a ◊ ◊ ◊ ◊ b
◊ ◊ a b	a ◊ ◊ b c ◊ ◊ d
a ◊ ◊ b ◊	a ◊ ◊ ◊ ◊ b ◊ c ◊ d
a ◊ ◊ b c	a ◊ ◊ b ◊ ◊ c ◊ ◊ d

Figure 9: A selection of rule patterns that are scope ≤ 3 but not LNF or CNF.

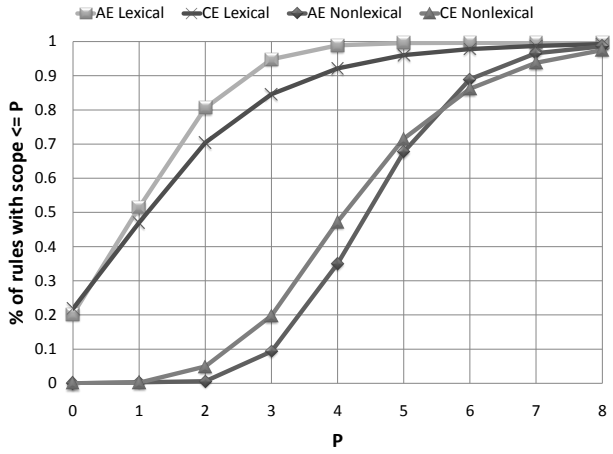

Figure 10: Breakdown of rules by scope (average per sentence in our test sets). In practice, most of the lexical rules applicable to a given sentence (95% for Arabic-English and 85% for Chinese-English) are scope 3 or less.

assert that this loss is negligable.

A drawback of their technique is that the resulting CNF grammar contains many more rules and postconditions than the original grammar. These constant factors do not impact asymptotic performance, but do impact practical performance.

7.2 Lexical Normal Form

Concerned about this inflation of the grammar constant, (DeNero et al., 2009) consider a superset of CNF called Lexical Normal Form (LNF). A rule is *LNF* if its pattern does not contain an ambiguity as a proper subsentence (recall that an ambiguity was defined to be the sentence "◊ ◊"). Like CNF, the maximum scope of an LNF grammar is 3. In the worst case, the pattern s^* is "◊ ◊", in which case there are three ambiguities in the sentence "◊" $\cdot s^* \cdot$ "◊".

(DeNero et al., 2009) provide a lossless scope reduction method that maps a simple grammar to an LNF grammar, thus enabling cubic-time decoding. Their principal objective is to provide a scope reduction method for SCFG that introduces fewer postconditions than (Zhang et al., 2006). However unlike (Zhang et al., 2006), their method *only* addresses simple grammars. Thus they cannot integrate LM scoring into their decoding, requiring them to rescore the decoder output with a variant of cube growing (Huang and Chiang, 2007).

7.3 Scope Pruning

To exercise the power of the ideas presented in this paper, we experimented with a third (and very easy) scope reduction method called *scope pruning*. If we consider the entire space of scope-3 grammars, we see that it contains a much richer set of rules than those permitted by CNF or LNF. See Figure 9 for examples. Scope pruning is a lossy scope reduction method that simply takes an arbitrary grammar and prunes all rules with scope greater than 3. By not modifying any rules, we preserve their cost and carry functions (enabling integrated LM decoding), without increasing the grammar constant. The practical question is: how many rules are we typically pruning from the original grammar?

We experimented with two pretrained syntax-based machine translation systems with rules extracted via the GHKM algorithm (Galley et al., 2004). The first was an Arabic-English system, with rules extracted from 200 million words of parallel data from the NIST 2008 data collection, and with a 4-gram language model trained on 1 billion words of monolingual English data from the LDC Gigaword corpus. We evaluated this system's performance on the NIST 2008 test corpus, which consists of 1357 Arabic sentences from a mixture of newswire and web domains, with four English reference translations. The second system was a Chinese-

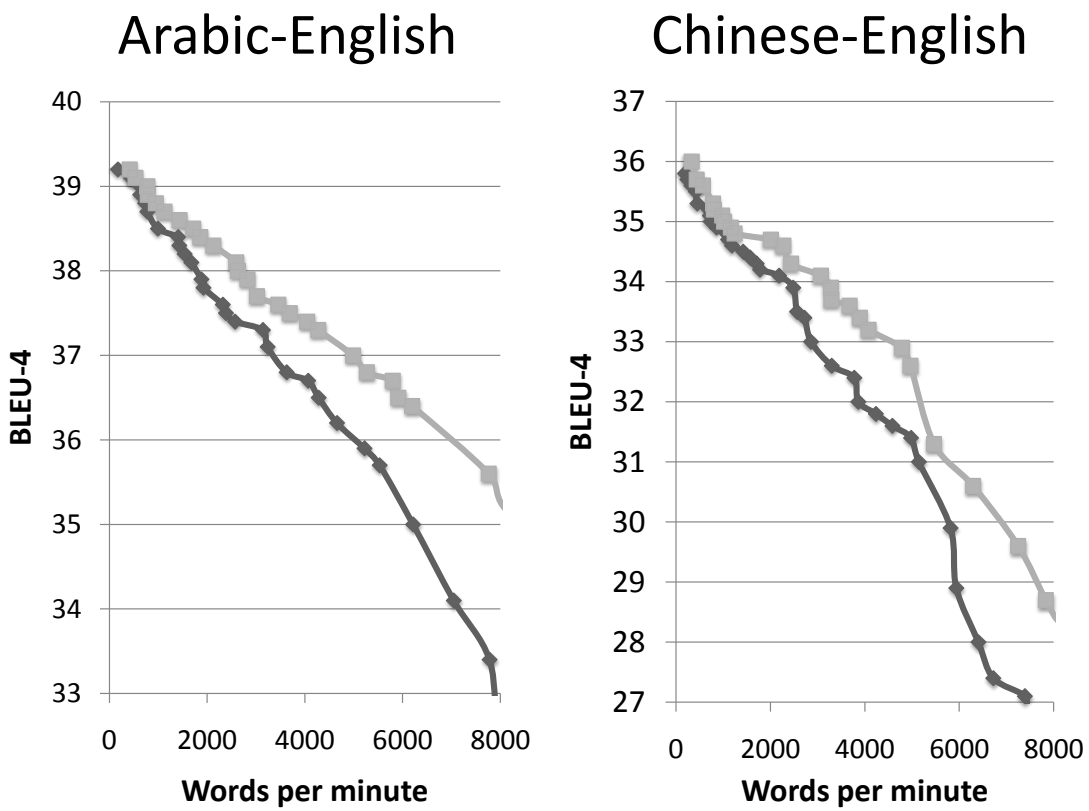

Figure 11: Speed-quality tradeoff curves comparing the baseline scope reduction method of synchronous binarization (dark gray diamonds) with scope-3 pruning (light gray squares).

English system, with rules extracted from 16 million words of parallel data from the mainland-news domain of the LDC corpora, and with a 4-gram language model trained on monolingual English data from the AFP and Xinhua portions of the LDC Gigaword corpus. We evaluated this system's performance on the NIST 2003 test corpus, which consists of 919 Chinese sentences, with four English reference translations. For both systems, we report BLEU scores (Papineni et al., 2002) on untokenized, recapitalized output.

In practice, how many rules have scope greater than 3? To answer this question, it is useful to distinguish between *lexical* rules (i.e. rules whose patterns contain at least one non-substitution symbol) and *non-lexical* rules. Only a subset of lexical rules are potentially applicable to a given input sentence. Figure 10 shows the scope profile of these applicable rules (averaged over all sentences in our test sets). Most of the lexical rules applicable to a given sentence (95% for Arabic-English, 85% for Chinese-English) are scope 3 or less. [8] Note, however, that scope pruning also prunes a large percentage of non-lexical rules.

Figure 11 compares scope pruning with the baseline technique of synchronous binarization. To generate these speed-quality tradeoff curves, we decoded the test sets with 380 different beam settings. We then plotted the hull of these 380 points, by eliminating any points that were dominated by another (i.e. had better speed and quality). We found that this simple approach to scope reduction produced a better speed-quality tradeoff than the much more complex synchronous binarization. [9]

[8] For contrast, the corresponding numbers for LNF are 64% and 53%, respectively.

[9] We also tried a hybrid approach in which we scope-pruned

8 Conclusion

In this paper, we made the following contributions:

- We provided an abstract formulation of chart parsing that generalizes CFG decoding and SCFG decoding with an integrated LM.
- We framed *scope reduction* as a first-class abstract problem, and showed that CNF binarization and LNF binarization are two specific solutions to this problem, each with their respective advantages and disadvantages.
- We proposed a third scope reduction technique called *scope pruning*, and we showed that it can outperform synchronous CNF binarization for particular use cases.

Moreover, this work gives formal expression to the extraction heuristics of hierarchical phrase-based translation (Chiang, 2007), whose directive not to extract SCFG rules with adjacent nonterminals can be viewed as a preemptive pruning of rules with scope greater than 2 (more specifically, the pruning of non-LNF lexical rules). In general, this work provides a framework in which different approaches to *tractability-focused* grammar construction can be compared and discussed.

References

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

John DeNero, Mohit Bansal, Adam Pauls, and Dan Klein. 2009. Efficient parsing for transducer grammars. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Proceedings of HLT/NAACL*.

Mark Hopkins and Greg Langmead. 2009. Cube pruning as heuristic search. In *Proceedings of EMNLP*.

Liang Huang and David Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of ACL*.

the lexical rules and synchronously binarized the non-lexical rules. This had a similar performance to scope-pruning all rules. The opposite approach of scope-pruning the lexical rules and synchronously binarizing the non-lexical rules had a similar performance to synchronous binarization.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318.

Daniel Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.

Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous binarization for machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 256–263.