Extracting Synchronous Grammar Rules From Word-Level Alignments in Linear Time

Hao Zhang and **Daniel Gildea**
Computer Science Department
University of Rochester
Rochester, NY 14627, USA

David Chiang
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292, USA

Abstract

We generalize Uno and Yagiura's algorithm for finding all common intervals of two permutations to the setting of two sequences with many-to-many alignment links across the two sides. We show how to maximally decompose a word-aligned sentence pair in linear time, which can be used to generate all possible phrase pairs or a Synchronous Context-Free Grammar (SCFG) with the simplest rules possible. We also use the algorithm to precisely analyze the maximum SCFG rule length needed to cover hand-aligned data from various language pairs.

1 Introduction

Many recent syntax-based statistical machine translation systems fall into the general formalism of Synchronous Context-Free Grammars (SCFG), where the grammar rules are found by first aligning parallel text at the word level. From word-level alignments, such systems extract the grammar rules consistent either with the alignments and parse trees for one of languages (Galley et al., 2004), or with the the word-level alignments alone without reference to external syntactic analysis (Chiang, 2005), which is the scenario we address here.

In this paper, we derive an optimal, linear-time algorithm for the problem of decomposing an arbitrary word-level alignment into SCFG rules such that each rule has at least one aligned word and is *minimal* in the sense that it cannot be further decomposed into smaller rules. Extracting minimal rules is of interest both because rules with fewer words are more likely to generalize to new data, and because rules with lower *rank* (the number of nonterminals on the right-hand side) can be parsed more efficiently.

This algorithm extends previous work on factoring permutations to the general case of factoring many-to-many alignments. Given two permutations of n, a *common interval* is a set of numbers that are consecutive in both. The breakthrough algorithm of Uno and Yagiura (2000) computes all K common intervals of two length n permutations in $O(n + K)$ time. This is achieved by designing data structures to index possible boundaries of common intervals as the computation proceeds, so that not all possible pairs of beginning and end points need to be considered. Landau et al. (2005) and Bui-Xuan et al. (2005) show that all common intervals can be encoded in $O(n)$ space, and adapt Uno and Yagiura's algorithm to produce this compact representation in $O(n)$ time. Zhang and Gildea (2007) use similar techniques to factorize Synchronous Context Free Grammars in linear time.

These previous algorithms assume that the input is a permutation, but in machine translation it is common to work with word-level alignments that are many-to-many; in general any set of pairs of words, one from each language, is a valid alignment for a given bilingual sentence pair. In this paper, we consider a generalized concept of common intervals given such an alignment: a common interval is a pair of phrases such that no word pair in the alignment links a word inside the phrase to a word outside the phrase. Extraction of such phrases is a common feature of state-of-the-art phrase-based and syntax-based machine translation systems (Och and Ney, 2004a; Chiang, 2005). We generalize Uno and Yagiura's algorithm to this setting, and demonstrate a linear time algorithm for a pair of aligned sequences. The output is a tree representation of possible phrases, which directly provides a set of minimal synchronous grammar

rules for an SCFG-based machine translation system. For phrase-based machine translation, one can also read all phrase pairs consistent with the original alignment off of the tree in time linear in the number of such phrases.

2 Alignments and Phrase Pairs

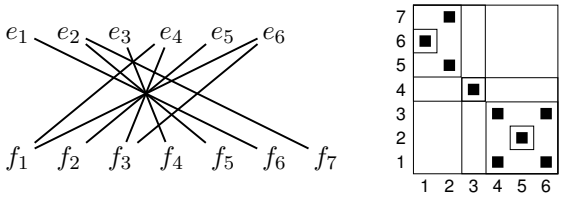
Let $[x, y]$ denote the sequence of integers between x and y inclusive, and $[x, y)$ the integers between x and $y - 1$ inclusive. An *aligned sequence pair* or simply an *alignment* is a tuple (E, F, A), where $E = e_1 \cdots e_n$ and $F = f_1 \cdots f_m$ are strings, and A is a set of links (x, y), where $1 \leq x \leq n$ and $1 \leq y \leq m$, connecting E and F. For most of this paper, since we are not concerned with the identity of the symbols in E and F, we will assume for simplicity that $e_i = i$ and $f_j = j$, so that $E = [1, n]$ and $F = [1, m]$.

In the context of statistical machine translation (Brown et al., 1993), we may interpret E as an English sentence, F its translation in French, and A a representation of how the words correspond to each other in the two sentences. A pair of substrings $[s, t] \subset E$ and $[u, v] \subset F$ is a *phrase pair* (Och and Ney, 2004b) if and only if the subset of links emitted from $[s, t]$ in E is equal to the subset of links emitted from $[u, v]$ in F, and both are nonempty.

Figure 1a shows an example of a many-to-many alignment, where $E = [1, 6]$, $F = [1, 7]$, and $A = \{(1, 6), (2, 5), (2, 7), (3, 4), (4, 1), (4, 3), (5, 2), (6, 1), (6, 3)\}$. The eight phrase pairs in this alignment are:

$$([1, 1],\ [6, 6]),\ ([1, 2],\ [5, 7]),$$
$$([3, 3],\ [4, 4]),\ ([1, 3],\ [4, 7]),$$
$$([5, 5],\ [2, 2]),\ ([4, 6],\ [1, 3]),$$
$$([3, 6],\ [1, 4]),\ ([1, 6],\ [1, 7]).$$

In Figure 1b, we show the *alignment matrix* representation of the given alignment. By default, the columns correspond to the tokens in E, the rows correspond to the tokens in F, and the black cells in the matrix are the alignment links in A. Using the matrix representation, the phrase pairs can be viewed as submatrices as shown with the black-lined boundary boxes. Visually, a submatrix represents a phrase pair when it contains at least one alignment link and there are no alignment links directly above, below, or to the right or left of it.



(a) (b)

Figure 1: An example of (a) a many-to-many alignment and (b) the same alignment as a matrix, with its phrase pairs marked.

2.1 Number of Phrase Pairs

In this section, we refine our definition of phrase pairs with the concept of *tightness* and give an asymptotic upper bound on the total number of such phrase pairs as the two sequences' lengths grow. In the original definition, the permissive many-to-many constraint allows for unaligned tokens in both sequences E and F. If there is an unaligned token adjacent to a phrase pair, then there is also a phrase pair that includes the unaligned token. We say that a phrase pair $([s, t], [u, v])$ is *tight* if none of e_s, e_t, f_u and f_v is unaligned. By focusing on tight phrase pairs, we eliminate the non-tight ones that share the same set of alignment links with their tight counterpart.

Given $[s, t]$ in E, let l be the first member of F that any position in $[s, t]$ links to, and let u be the last. According to the definition of tight phrase pair, $[l, u]$ is the only candidate phrase in F to pair up with $[s, t]$ in E. So, the total number of tight phrase pairs is upper-bounded by the total number of intervals in each sequence, which is $O(n^2)$.

If we do not enforce the tightness constraint, the total number of phrase pairs can grow much faster. For example, if a sentence contains only a single alignment link between the midpoint of F and the midpoint of E, then there will be $O(n^2m^2)$ possible phrase pairs, but only a single tight phrase pair. From now on, term *phrase pair* always refers to a tight phrase pair.

2.2 Hierarchical Decomposition of Phrase Pairs

In this section, we show how to encode all the tight phrase pairs of an alignment in a tree of size $O(n)$.

Lemma 2.1. *When two phrase pairs overlap, the intersection, the differences, and the union of the two are also phrase pairs.*

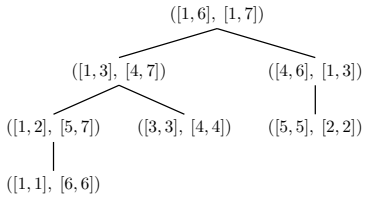
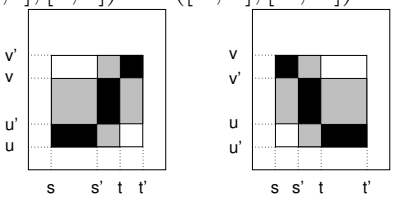


Figure 2: The normalized decomposition tree of the alignment in Figure 1.

The following picture graphically represents the two possible overlapping structures of two phrase pairs: $([s,t],[u,v])$ and $([s',t'],[u',v'])$.



Let AB and BC be two overlapping English phrases, with B being their overlap. There are six possible phrases, A, B, C, AB, BC, and ABC, but if we omit BC, the remainder are nested and can be represented compactly by $((AB)C)$, from which BC can easily be recovered. If we systematically apply this to the whole sentence, we obtain a hierarchical representation of all the phrase pairs, which we call the *normalized decomposition tree*. The normalized decomposition tree for the example is shown in Figure 2.

Bui-Xuan et al. (2005) show that the family of common intervals is *weakly partitive*, i.e. closed under intersection, difference and union. This allows the family to be represented as a hierarchical decomposition. The normalized decomposition focuses on the *right strong intervals*, those that do not overlap with any others on the right. Lemma 2.1 shows that the family of phrase pairs is also a weakly partitive family and can be hierarchically decomposed after normalization. A minor difference is we prefer *left strong intervals* since our algorithms scan F from left to right. Another difference is that we binarize a linearly-arranged sequence of non-overlapping phrase pairs instead of grouping them together.

In the following sections, we show how to produce the normalized hierarchical analysis of a given alignment.

3 Shift-Reduce Algorithm

In this section, we present an $O(n^2 + m + |A|)$ algorithm that is similar in spirit to a shift-reduce algorithm for parsing context-free languages. This algorithm is not optimal, but its left-to-right bottom-up control will form the basis for the improved algorithm in the next section.

First, we can efficiently test whether a span $[x,y]$ is a phrase as follows. Define a pair of functions $l(x,y)$ and $u(x,y)$ that record the minimum and maximum, respectively, of the positions on the French side that are linked to the positions $[x,y]$:

$$l(x,y) = \min\{j \mid (i,j) \in A, i \in [x,y]\}$$
$$u(x,y) = \max\{j \mid (i,j) \in A, i \in [x,y]\}$$

Note that $l(\cdot,y)$ is monotone increasing and $u(\cdot,y)$ is monotone decreasing. Define a *step* of $l(\cdot,y)$ (or $u(\cdot,y)$) to be a maximal interval over which $l(\cdot,y)$ (resp., $u(\cdot,y)$) is constant. We can compute $u(x,y)$ in constant time from its value on smaller spans:

$$u(x,y) = \max\{u(x,z), u(z+1,y)\}$$

and similarly for $l(x,y)$.

We define the following functions to count the number of links emitted from prefixes of F and E:

$$F_c(j) = |\{(i',j') \in A \mid j' \leq j\}|$$
$$E_c(i) = |\{(i',j') \in A \mid i' \leq i\}|$$

Then the difference $F_c(u) - F_c(l-1)$ counts the total number of links to positions in $[l,u]$, and $E_c(y) - E_c(x-1)$ counts the total number of links to positions in $[x,y]$. E_c and F_c can be precomputed in $O(n+m+|A|)$ time.

Finally, let

$$f(x,y) = F_c(u(x,y)) - F_c(l(x,y)-1) - (E_c(y) - E_c(x-1))$$

Note that f is non-negative, but not monotonic in general. Figure 4 provides a visualization of u, l, and f for the example alignment from Section 2. This gives us our phrase-pair test:

Lemma 3.1. *$[x,y]$ and $[l(x,y),u(x,y)]$ are a phrase pair if and only if $f(x,y) = 0$.*

This test is used in the following shift-reduce-style algorithm:

$X \leftarrow \{1\}$

for $y \in [2, n]$ from left to right **do**
  append y to X
  for $x \in X$ from right to left **do**
    compute $u(x, y)$ from $u(x + 1, y)$
    compute $l(x, y)$ from $l(x + 1, y)$
    if $f(x, y) = 0$ **then**
      $[x, y]$ is a phrase
      remove $[x + 1, y]$ from X
    end if
  end for
end for

In the worst case, at each iteration we traverse the entire stack X without a successful reduction, indicating that the worst case time complexity is $O(n^2)$.

4 A Linear Algorithm

In this section, we modify the shift-reduce algorithm into a linear-time algorithm that avoids unnecessary reduction attempts. It is a generalization of Uno and Yagiura's algorithm.

4.1 Motivation

The reason that our previous algorithm is quadratic is that for each y, we try every possible combination with the values in X. Uno and Yagiura (2000) point out that in the case of permutations, it is not necessary to examine all spans, because it is possible to delete elements from X so that $f(\cdot, y)$ is monotone decreasing on X. This means that all the $x \in X$ such that $f(x, y) = 0$ can always be conveniently found at the end of X. That this can be done safely is guaranteed by the following:

Lemma 4.1. *If $x_1 < x_2 < y$ and $f(x_1, y) < f(x_2, y)$, then for all $y' \geq y$, $f(x_2, y') > 0$ (i.e., $[x_2, y']$ is not a phrase).*

Let us say that x_2 *violates monotonicity* if x_1 is the predecessor of x_2 in X and $f(x_1, y) < f(x_2, y)$. Then by Lemma 4.1, we can safely remove x_2 from X.

Furthermore, Uno and Yagiura (2000) show that we can enforce monotonicity at all times in such a way that the whole algorithm runs in linear time. This is made possible with a shortcut based on the following:

Lemma 4.2. *If $x_1 < x_2 < y$ and $u(x_1, y - 1) > u(x_2, y - 1)$ but $u(x_1, y) = u(x_2, y)$, then for all $y' \geq y$, $f(x_2, y') > 0$ (i.e., $[x_2, y']$ is not a phrase). The same holds* mutatis mutandis *for l.*

Let us say that y *updates* a step $[x', y']$ of u (or l) if $u(x', y) > u(x', y - 1)$ (resp., $l(x', y) < l(x', y-1)$). By Lemma 4.2, if $[x_1, y_1]$ and $[x_2, y_2]$ are different steps of $u(\cdot, y - 1)$ (resp., $l(\cdot, y - 1)$) and y updates both of them, then we can remove from X all x' such that $x_2 \leq x' < y$.

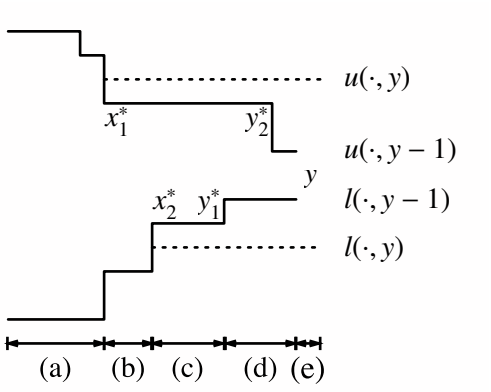


Figure 3: Illustration of step (3) of the algorithm. The letters indicate substeps of (3).

4.2 Generalized algorithm

These results generalize to the many-to-many alignment case, although we must introduce a few nuances. The new algorithm proceeds as follows:

Initialize $X = \{1\}$. For $y \in [2, n]$ from left to right:

1. Append y to X.
2. Update u and l:
   (a) Traverse the steps of $u(\cdot, y - 1)$ from right to left and compute $u(\cdot, y)$ until we have found the leftmost step $[x^*, y^*]$ of $u(\cdot, y - 1)$ that gets updated by y.
   (b) Do the same for l.

   We have computed two values for x^*; let x_1^* be the smaller and x_2^* be the larger. Similarly, let y_1^* be the smaller y^*.
3. Enforce monotonicity of $f(\cdot, y)$ (see Figure 3):
   (a) The positions left of the smaller x^* always satisfy monotonicity, so do nothing.
   (b) For $x \in [x_1^*, x_2^*) \cap X$ while x violates monotonicity, remove x from X.

(c) For $x \in [x_2^*, y_1^*] \cap X$ while x violates monotonicity, remove x from X.

(d) The steps right of y_1^* may or may not violate monotonicity, but we use the stronger Lemma 4.2 to delete all of them (excluding y).[1]

(e) Finally, if y violates monotonicity, remove it from X.

4. For $x \in X$ from right to left until $f(x, y) > 0$, output $[x, y]$ and remove x's successor in X.[2]

An example of the algorithm's execution is shown in Figure 4. The evolution of $u(x, y)$, $l(x, y)$, and $f(x, y)$ is displayed for increasing y (from 1 to 6). We point out the interesting steps. When $y = 2$, position 2 is eliminated due to step (3e) of our algorithm to ensure monotonicity of f at the right end, and $[1, 2]$ is reduced. When $y = 3$, two reductions are made: one on $[3, 3]$ and the other on $[1, 3]$. Because of leftmost normalization, position 3 is deleted. When $y = 6$, we have $x_1^* = x_2^* = y_1^* = 5$, so that position 5 is deleted by step (3c) and position 6 is deleted by step (3e).

4.3 Correctness

We have already argued in Section 4.1 that the deletion of elements from X does not alter the output of the algorithm. It remains to show that step (3) guarantees monotonicity:

Claim 4.3. *For all y, at the end of step (3), $f(\cdot, y)$ is monotone decreasing.*

Proof. By induction on y. For $y = 1$, the claim is trivially true. For $y > 1$, we want to show that for x_1, x_2 adjacent in X such that $x_1 < x_2$, $f(x_1, y) \geq f(x_2, y)$. We consider the five regions of X covered by step (3) (cf. Figure 3), and then the boundaries between them.

Region (a): $x_1, x_2 \in [1, x_1^*]$. Since $u(x_i, y) = u(x_i, y - 1)$ and $l(x_i, y) = l(x_i, y - 1)$, we have:

$$f(x_i, y) - f(x_i, y-1) = 0 - (E_c(y) - E_c(y-1))$$

i.e., in this region, f shifts down uniformly from iteration $y - 1$ to iteration y. Hence, if $f(\cdot, y - 1)$ was monotonic, then $f(\cdot, y)$ is also monotonic within this region.

Region (b): $x_1, x_2 \in [x_1^*, x_2^*)$. Since $u(x_1, y - 1) = u(x_2, y - 1)$ and $u(x_1, y) = u(x_2, y)$ and similarly for l, we have:

$$f(x_1, y) - f(x_1, y-1) = f(x_2, y) - f(x_2, y-1)$$

i.e., in this region, f shifts up or down uniformly.[3] Hence, if $f(\cdot, y - 1)$ was monotonic, then $f(\cdot, y)$ is also monotonic within this region.

Region (c): $x_1, x_2 \in [x_2^*, y_1^*]$. Same as Case 2.

Region (d) and (e): Vacuous (these regions have at most one element).

The remaining values of x_1, x_2 are those that straddle the boundaries between regions. But step (3) of the algorithm deals with each of these boundaries explicitly, deleting elements until $f(x_1) \geq f(x_2)$. Thus $f(\cdot, y)$ is monotonic everywhere. □

4.4 Implementation and running time

X should be implemented in a way that allows linear-time traversal and constant-time deletion; also, u and l must be implemented in a way that allows linear-time traversal of their steps. Doubly-linked lists are appropriate for all three functions.

Claim 4.4. *The above algorithm runs in $O(n + m + |A|)$ time.*

We can see that the algorithm runs in linear time if we observe that whenever we traverse a part of X, we delete it, except for a constant amount of work per iteration (that is, per value of y): the steps traversed in (2) are all deleted in (3d) except four (two for u and two for l); the positions traversed in (3b), (3c), and (4) are all deleted except one.

4.5 SCFG Rule extraction

The algorithm of the previous section outputs the normalized decomposition tree depicted in Figure 2. From this tree, it is straightforward to obtain a set of maximally-decomposed SCFG rules. As an example, the tree of Figure 2 produces the rules shown in Figure 5.

[1] In the special case where $[x^*, y^*]$ is updated by y to the same value as the step to its left, we can use Lemma 4.2 to delete $[x^*, y^*]$ and y as well, bypassing steps (3b),(3c), and (3e).

[2] If there are any such x, they must lie to the left of x_1^*. Therefore a further optimization would be to perform step (4) before step (3), starting with the predecessor of x_1^*. If a reduction is made, we can jump to step (3e).

[3] It can be shown further that in this region, f shifts up or is unchanged. Therefore any reductions in step (4) must be in region (a).

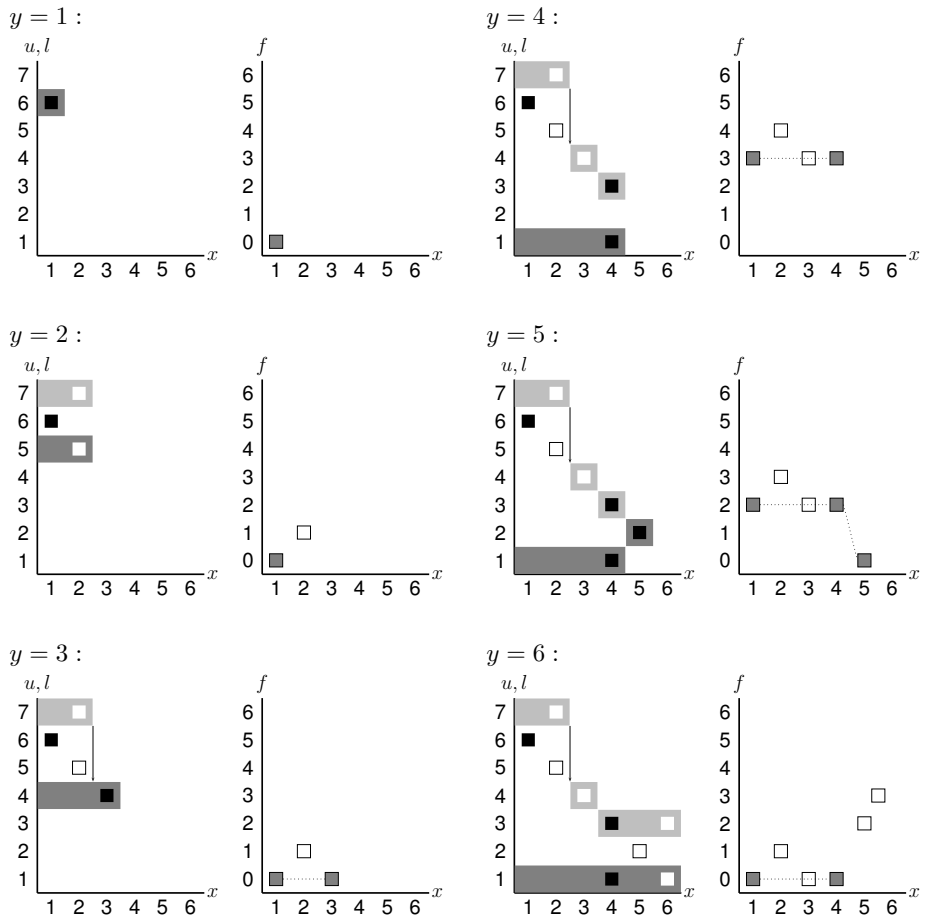


Figure 4: The evolution of $u(x, y)$, $l(x, y)$, and $f(x, y)$ as y goes from 1 to 6 for the example alignment. Each pair of diagrams shows the state of affairs between steps (3) and (4) of the algorithm. Light grey boxes are the steps of u, and darker grey boxes are the steps of l. We use solid boxes to plot the values of remaining x's on the list but also show the other values in empty boxes for completeness.

We adopt the SCFG notation of Satta and Peserico (2005). Each rule has a right-hand side sequence for both languages, separated by a comma. Superscript indices in the right-hand side of grammar rules such as:

$$A \rightarrow B^{(1)}\, C^{(2)},\ C^{(2)}\, B^{(1)}$$

indicate that the nonterminals with the same index are linked across the two languages, and will eventually be rewritten by the same rule application. The example above inverts the order of B and C when translating from the source language to the target language.

The SCFG rule extraction proceeds as follows. Assign a nonterminal label to each node in the tree. Then for each node (S, T) in the tree top-down, where S and T are sequences of positions,

1. For each child (S', T'), S' and T' must be subsequences of S and T, respectively. Replace their occurrences in S and T with a pair of coindexed nonterminals X', where X' is the nonterminal assigned to the child.

2. For each remaining position i in S, replace i with e_i.

3. For each remaining position j in T, replace j with f_j.

4. Output the rule $X \rightarrow S,\ T$, where X is the nonterminal assigned to the parent.

As an example, consider the node $([4, 6], [1, 3])$ in Figure 2. After step 1, it becomes

$$(4\, F^{(1)}\, 6, 1\, F^{(1)}\, 3)$$

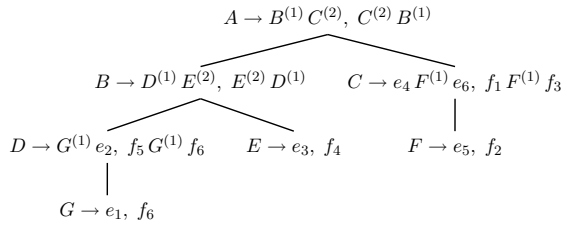


Figure 5: Each node from the normalized decomposition tree of Figure 2 is converted into an SCFG rule.

and after steps 2 and 3, it becomes

$$(e_4\, F^{(1)}\, e_6, f_1\, F^{(1)}\, f_3)$$

Finally, step 4 outputs

$$C \rightarrow e_4\, F^{(1)}\, e_6,\ f_1\, F^{(1)}\, f_3$$

A few choices are available to the user depending on the application intended for the SCFG extraction. The above algorithm starts by assigning a nonterminal to each node in the decomposition tree; one could assign a unique nonterminal to each node, so that the resulting grammar produces exactly the set of sentences given as input. But for machine translation, one may wish to use a single nonterminal, such that the extracted rules can recombine freely, as in Chiang (2005).

Unaligned words in either language (an empty row or column in the alignment matrix, not present in our example) will be attached as high as possible in our tree. However, other ways of handling unaligned words are possible given the decomposition tree. One can produce all SCFG rules consistent with the alignment by, for each unaligned word, looping through possible attachment points in the decomposition tree. In this case, the number of SCFG rules produced may be exponential in the size of the original input sentence; however, even in this case, the decomposition tree enables a rule extraction algorithm that is linear in the output length (the number of SCFG rules).

4.6 Phrase extraction

We briefly discuss the process of extracting all phrase pairs consistent with the original alignment from the normalized decomposition tree. First of all, every node in the tree gives a valid phrase pair. Then, in the case of overlapping phrase pairs such as the example in Section 2.1, the decomposition tree will contain a left-branching chain of binary nodes all performing the same permutation. While traversing the tree, whenever we identify such a chain, let $\eta_1, \ldots, \eta_k$ be the sequence of all the children of the nodes in the chain. Then, each of the subsequences $\{\eta_i, \ldots, \eta_j \mid 1 < i < j \leq k\}$ yields a valid phrase pair. In our example, the root of the tree of Figure 2 and its left child form such a chain, with three children; the subsequence $\{([3,3],[4,4]),([4,6],[1,3])\}$ yields the phrase $([3,6],[1,4])$. In the case of unaligned words, we can also consider all combinations of their attachments, as discussed for SCFG rule extraction.

5 Experiments on Analyzing Word Alignments

One application of our factorization algorithm is analyzing human-annotated word alignments. Wellington et al. (2006) argue for the necessity of discontinuous spans (i.e., for a formalism beyond Synchronous CFG) in order for synchronous parsing to cover human-annotated word alignment data under the constraint that rules have a rank of no more than two. In a related study, Zhang and Gildea (2007) analyze the rank of the Synchronous CFG derivation trees needed to parse the same data. The number of discontinuous spans and the rank determine the complexity of dynamic programming algorithms for synchronous parsing (alignment) or machine translation decoding.

Both studies make simplifying assumptions on the alignment data to avoid dealing with many-to-many word links. Here, we apply our alignment factorization algorithm directly to the alignments to produce a normalized decomposition tree for each alignment and collect statistics on the branching factors of the trees.

We use the same alignment data for the five language pairs Chinese-English, Romanian-English, Hindi-English, Spanish-English, and French-English as Wellington et al. (2006). Table 1 reports the number of rules extracted by the rank, or number of nonterminals on the right-hand side. Almost all rules are binary, implying both that binary synchronous grammars are adequate for MT, and that our algorithm can find such grammars. Table 2 gives similar statistics for the number of *terminals* in each rule. The phrases we extract are short enough that they are likely to generalize to new sentences. The apparent difficulty of

	0	1	2	3	4	5	6
Hindi/English	52.8	53.5	99.9		99.9	100.0	
Chinese/English	51.0	52.4	99.7	99.8	100.0	100.0	100.0
French/English	52.1	53.5	99.9	100.0	100.0	100.0	
Romanian/English	50.8	52.6	99.9	99.9	100.0		100.0
Spanish/English	50.7	51.8	99.9	100.0	100.0	100.0	

Table 1: Cumulative percentages of rule tokens by number of nonterminals in right-hand side. A blank indicates that no rules were found with that number of nonterminals.

	0	1	2	3	4	5	6	7	8	9	≥10	max
Hindi/English	39.6	92.2	97.7	99.5	99.7	99.9	99.9	100.0				7
Chinese/English	39.8	87.2	96.2	99.0	99.7	99.9	100.0	100.0	100.0	100.0	100.0	12
French/English	44.5	89.0	93.4	95.8	97.5	98.4	99.0	99.3	99.6	99.8	100.0	18
Romanian/English	42.9	89.8	96.9	98.9	99.5	99.8	99.9	100.0		100.0		9
Spanish/English	47.5	91.8	97.7	99.4	99.9	99.9	100.0	100.0		100.0		9

Table 2: Cumulative percentages of rule tokens by number of terminals in right-hand side. A blank indicates that no rules were found with that number of terminals.

the French-English pair is due to the large number of "possible" alignments in this dataset.

6 Conclusion

By extending the algorithm of Uno and Yagiura (2000) from one-to-one mappings to many-to-many mappings, we have shown how to construct a hierarchical representation of all the phrase pairs in a given aligned sentence pair in linear time, which yields a set of minimal SCFG rules. We have also illustrated how to apply the algorithm as an analytical tool for aligned bilingual data.

Acknowledgments Thanks to Bob Moore for suggesting the extension to phrase extraction at SSST 2007. This work was supported in part by NSF grants IIS-0546554 and ITR-0428020, and DARPA grant HR0011-06-C-0022 under BBN Technologies subcontract 9500008412.

References

Brown, Peter F., Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Bui-Xuan, Binh Minh, Michel Habib, and Christophe Paul. 2005. Revisiting T. Uno and M. Yagiura's algorithm. In *The 16th Annual International Symposium on Algorithms and Computation (ISAAC '05)*, pages 146–155.

Chiang, David. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of ACL 2005*, pages 263–270.

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

Landau, Gad M., Laxmi Parida, and Oren Weimann. 2005. Gene proximity analysis across whole genomes via PQ trees. *Journal of Computational Biology*, 12(10):1289–1306.

Och, Franz Josef and Hermann Ney. 2004a. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4).

Och, Franz Josef and Hermann Ney. 2004b. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30:417–449.

Satta, Giorgio and Enoch Peserico. 2005. Some computational complexity results for synchronous context-free grammars. In *Proceedings of EMNLP 2005*, pages 803–810, Vancouver, Canada, October.

Uno, Takeaki and Mutsunori Yagiura. 2000. Fast algorithms to enumerate all common intervals of two permutations. *Algorithmica*, 26(2):290–309.

Wellington, Benjamin, Sonjia Waxmonsky, and I. Dan Melamed. 2006. Empirical lower bounds on the complexity of translational equivalence. In *Proceedings of COLING-ACL 2006*.

Zhang, Hao and Daniel Gildea. 2007. Factorization of synchronous context-free grammars in linear time. In *Proceedings of the NAACL Workshop on Syntax and Structure in Statistical Translation (SSST)*.