Worst-Case Synchronous Grammar Rules

Daniel Gildea and **Daniel Štefankovič**
Computer Science Dept.
University of Rochester
Rochester, NY 14627

Abstract

We relate the problem of finding the best application of a Synchronous Context-Free Grammar (SCFG) rule during parsing to a Markov Random Field. This representation allows us to use the theory of expander graphs to show that the complexity of SCFG parsing of an input sentence of length N is $\Omega(N^{cn})$, for a grammar with maximum rule length n and some constant c. This improves on the previous best result of $\Omega(N^{c\sqrt{n}})$.

1 Introduction

Recent interest in syntax-based methods for statistical machine translation has lead to work in parsing algorithms for synchronous context-free grammars (SCFGs). Generally, parsing complexity depends on the length of the longest rule in the grammar, but the exact nature of this relationship has only recently begun to be explored. It has been known since the early days of automata theory (Aho and Ullman, 1972) that the languages of string pairs generated by a synchronous grammar can be arranged in an infinite hierarchy, with each rule size ≥ 4 producing languages not possible with grammars restricted to smaller rules. For any grammar with maximum rule size n, a fairly straightforward dynamic programming strategy yields an $O(N^{n+4})$ algorithm for parsing sentences of length N. However, this is often not the best achievable complexity, and the exact bounds of the best possible algorithms are not known. Satta and Peserico (2005) showed that a permutation can be defined for any length n such that tabular parsing strategies must take at least $\Omega(N^{c\sqrt{n}})$, that is, the exponent of the algorithm is proportional to the square root of the rule length. In this paper, we improve this result, showing that in the worst case the exponent grows linearly with the rule length. Using a probabilistic argument, we show that the number of easily parsable permutations grows slowly enough that most permutations must be difficult, where by difficult we mean that the exponent in the complexity is greater than a constant factor times the rule length. Thus, not only do there exist permutations that have complexity higher than the square root case of Satta and Peserico (2005), but in fact the probability that a randomly chosen permutation will have higher complexity approaches one as the rule length grows.

Our approach is to first relate the problem of finding an efficient parsing algorithm to finding the *treewidth* of a graph derived from the SCFG rule's permutation. We then show that this class of graphs are *expander graphs*, which in turn means that the treewidth grows linearly with the graph size.

2 Synchronous Parsing Strategies

We write SCFG rules as productions with one lefthand side nonterminal and two righthand side strings. Nonterminals in the two strings are linked with superscript indices; symbols with the same index must be further rewritten synchronously. For example,

$$X \rightarrow A^{(1)}\, B^{(2)}\, C^{(3)}\, D^{(4)},\ A^{(1)}\, B^{(2)}\, C^{(3)}\, D^{(4)} \tag{1}$$

is a rule with four children and no reordering, while

$$X \rightarrow A^{(1)}\, B^{(2)}\, C^{(3)}\, D^{(4)},\ B^{(2)}\, D^{(4)}\, A^{(1)}\, C^{(3)} \tag{2}$$

Algorithm 1 BottomUpParser(grammar G, input strings $\mathbf{e}$, $\mathbf{f}$)

for x_0, x_n such that $1 < x_0 < x_n < |\mathbf{e}|$ in increasing order of $x_n - x_0$ **do**
  for y_0, y_n such that $1 < y_0 < y_n < |\mathbf{f}|$ in increasing order of $y_n - y_0$ **do**
    for Rules R of form $X \rightarrow X_1^{(1)}...X_n^{(n)},\ X_{\pi(1)}^{(\pi(1))}...X_{\pi(n)}^{(\pi(n))}$ in G **do**
      $p = P(R) \max_{\substack{x_1..x_{n-1} \\ y_1..y_{n-1}}} \prod_i \delta(X_i, x_{i-1}, x_i, y_{\pi(i)-1}, y_{\pi(i)})$
      $\delta(X, x_0, x_n, y_0, y_n) = \max\{\delta(X, x_0, x_n, y_0, y_n), p\}$
    end for
  end for
end for

expresses a more complex reordering. In general, we can take indices in the first grammar dimension to be consecutive, and associate a permutation π with the second dimension. If we use X_i for $0 \leq i \leq n$ as a set of variables over nonterminal symbols (for example, X_1 and X_2 may both stand for nonterminal A), we can write rules in the general form:

$$X_0 \rightarrow X_1^{(1)}...X_n^{(n)},\ X_{\pi(1)}^{(\pi(1))}...X_{\pi(n)}^{(\pi(n))}$$

Grammar rules also contain terminal symbols, but as their position does not affect parsing complexity, we focus on nonterminals and their associated permutation π in the remainder of the paper. In a probabilistic grammar, each rule R has an associated probability $P(R)$. The synchronous parsing problem consists of finding the tree covering both strings having the maximum product of rule probabilities.[1]

We assume synchronous parsing is done by storing a dynamic programming table of recognized nonterminals, as outlined in Algorithm 1. We refer to a dynamic programming item for a given nonterminal with specified boundaries in each language as a *cell*. The algorithm computes cells by maximizing over *boundary variables* x_i and y_i, which range over positions in the two input strings, and specify beginning and end points for the SCFG rule's child nonterminals.

The maximization in the inner loop of Algorithm 1 is the most expensive part of the procedure, as it would take $O(N^{2n-2})$ with exhaustive search; making this step more efficient is our focus in this paper. The maximization can be done with further dynamic programming, storing partial results which contain some subset of an SCFG rule's righthand side nonterminals that have been recognized. A parsing strategy for a specific SCFG rule consists of an order in which these subsets should be combined, until all the rule's children have been recognized. The complexity of an individual parsing step depends on the number of free boundary variables, each of which can take $O(N)$ values. It is often helpful to visualize parsing strategies on the *permutation matrix* corresponding to a rule's permutation π. Figure 1 shows the permutation matrix of rule (2) with a three-step parsing strategy. Each panel shows one combination step along with the projections of the partial results in each dimension; the endpoints of these projections correspond to free boundary variables. The second step has the highest number of distinct endpoints, five in the vertical dimension and three horizontally, meaning parsing can be done in time $O(N^8)$.

As an example of the impact that the choice of parsing strategy can make, Figure 2 shows a permutation for which a clever ordering of partial results enables parsing in time $O(N^{10})$ in the length of the input strings. Permutations having this pattern of diagonal stripes can be parsed using this strategy in time $O(N^{10})$ regardless of the length n of the SCFG rule, whereas a naïve strategy proceeding from left to right in either input string would take time $O(N^{n+3})$.

2.1 Markov Random Fields for Cells

In this section, we connect the maximization of probabilities for a cell to the Markov Random Field

[1] We describe our methods in terms of the Viterbi algorithm (using the max-product semiring), but they also apply to non-probabilistic parsing (boolean semiring), language modeling (sum-product semiring), and Expectation Maximization (with inside and outside passes).

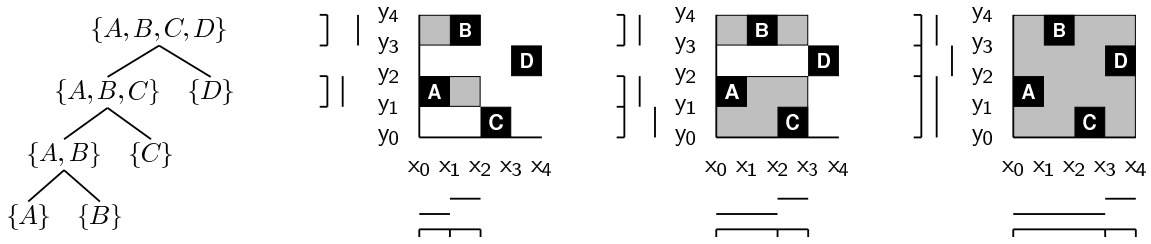

Figure 1: The tree on the left defines a three-step parsing strategy for rule (2). In each step, the two subsets of nonterminals in the inner marked spans are combined into a new chart item with the outer spans. The intersection of the outer spans, shaded, has now been processed. Tic marks indicate distinct endpoints of the spans being combined, corresponding to the free boundary variables.

(MRF) representation, which will later allow us to use algorithms and complexity results based on the graphical structure of MRFs. A Markov Random Field is defined as a probability distribution[2] over a set of variables $\mathbf{x}$ that can be written as a product of *factors* f_i that are functions of various subsets $\mathbf{x}_i$ of $\mathbf{x}$. The probability of an SCFG rule instance computed by Algorithm 1 can be written in this functional form:

$$\delta^R(\mathbf{x}) = P(R) \prod_i f_i(\mathbf{x}_i)$$

where

$$\mathbf{x} = \{x_i, y_i\} \text{ for } 0 \leq i \leq n$$

$$\mathbf{x}_i = \{x_{i-1}, x_i, y_{\pi(i)-1}, y_{\pi(i)}\}$$

and the MRF has one factor f_i for each child nonterminal X_i in the grammar rule R. The factor's value is the probability of the child nonterminal, which can be expressed as a function of its four boundaries:

$$f_i(\mathbf{x}_i) \quad = \quad \delta(X_i, x_{i-1}, x_i, y_{\pi(i)-1}, y_{\pi(i)})$$

For reasons that are explained in the following section, we augment our Markov Random Fields with a dummy factor for the completed parent nonterminal's chart item. Thus there is one dummy factor d for each grammar rule:

$$d(x_0, x_n, y_0, y_n) = 1$$

expressed as a function of the four *outer boundary variables* of the completed rule, but with a constant value of 1 so as not to change the probabilities computed.

[2] In our case unnormalized.

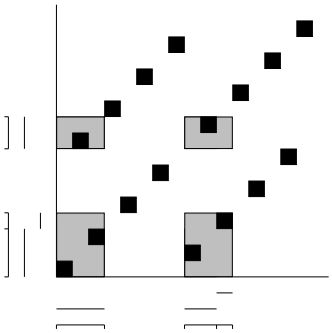

Figure 2: A parsing strategy maintaining two spans in each dimension is $O(N^{10})$ for any length permutation of this general form.

Thus an SCFG rule with n child nonterminals always results in a Markov Random Field with $2n+2$ variables and $n+1$ factors, with each factor a function of exactly four variables.

Markov Random Fields are often represented as graphs. A *factor graph* representation has a node for each variable and factor, with an edge connecting each factor to the variables it depends on. An example for rule (2) is shown in Figure 3, with round nodes for variables, square nodes for factors, and a diamond for the special dummy factor.

2.2 Junction Trees

Efficient computation on Markov Random Fields is performed by first transforming the MRF into a *junction tree* (Jensen et al., 1990; Shafer and Shenoy, 1990), and then applying the standard

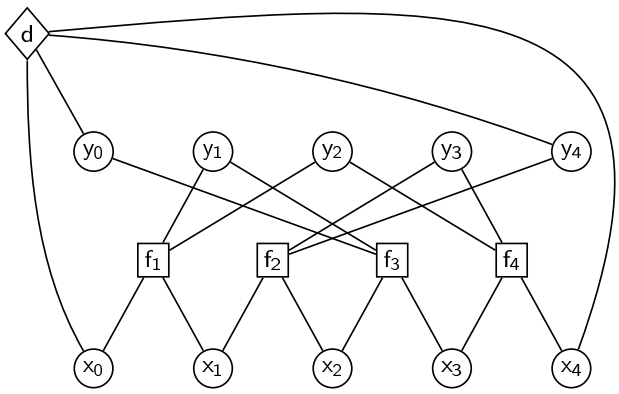

Figure 3: Markov Random Field for rule (2).

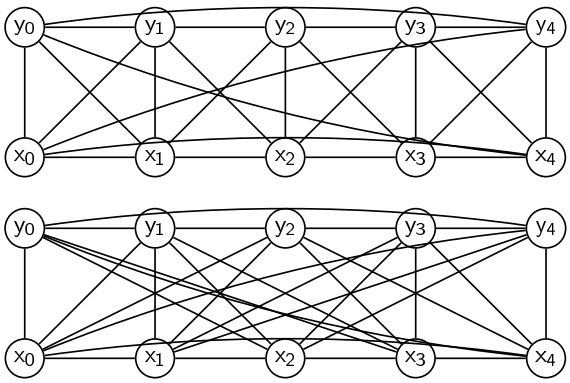

Figure 4: The graphs resulting from connecting all interacting variables for the identity permutation $(1, 2, 3, 4)$ (top) and the $(2, 4, 1, 3)$ permutation of rule (2) (bottom).

message-passing algorithm for graphical models over this tree structure. The complexity of the message passing algorithm depends on the structure of the junction tree, which in turn depends on the graph structure of the original MRF.

A junction tree can be constructed from a Markov Random Field by the following three steps:

- Connect all variable nodes that share a factor, and remove factor nodes. This results in the graphs shown in Figure 4.
- Choose a *triangulation* of the resulting graph, by adding chords to any cycle of length greater than three.
- Decompose the triangulated graph into a tree of cliques.

We call nodes in the resulting tree, corresponding to cliques in the triangulated graph, *clusters*. Each cluster has a *potential function*, which is a function of the variables in the cluster. For each factor in the original MRF, the junction tree will have at least one cluster containing all of the variables on which the factor is defined. Each factor is associated with one such cluster, and the cluster's potential function is set to be the product of its factors, for all combinations of variable values. Triangulation ensures that the resulting tree satisfies the *junction tree property*, which states that for any two clusters containing the same variable x, all nodes on the path connecting the clusters also contain x. A junction tree derived from the MRF of Figure 3 is shown in Figure 5.

The message-passing algorithm for graphical models can be applied to the junction tree. The algorithm works from the leaves of the tree inward, alternately multiplying in potential functions and maximizing over variables that are no longer needed, effectively distributing the $\max$ and product operators so as to minimize the interaction between variables. The complexity of the message-passing is $O(nN^k)$, where the junction tree contain $O(n)$ clusters, k is the maximum cluster size, and each variable in the cluster can take N values.

However, the standard algorithm assumes that the factor functions are predefined as part of the input. In our case, however, the factor functions themselves depend on message-passing calculations from other grammar rules:

$$f_i(\mathbf{x}_i) = \delta(X_i, x_{i-1}, x_i, y_{\pi(i)-1}, y_{\pi(i)})$$
$$= \max_{R' : X_i \to \alpha,\beta} P(R') \max_{\substack{\mathbf{x}': \\ x'_0 = x_{i-1}, x'_{n'} = x_i \\ y'_0 = y_{\pi(i-1)}, y'_{n'} = y_{\pi(i)}}} \delta^{R'}(\mathbf{x}') \quad (3)$$

We must modify the standard algorithm in order to interleave computation among the junction trees corresponding to the various rules in the grammar, using the bottom-up ordering of computation from Algorithm 1. Where, in the standard algorithm, each message contains a complete table for all assignments to its variables, we break these into a separate message for each individual assignment of variables. The overall complexity is unchanged, because each assignment to all variables in each cluster is still considered only once.

The dummy factor d ensures that every junction

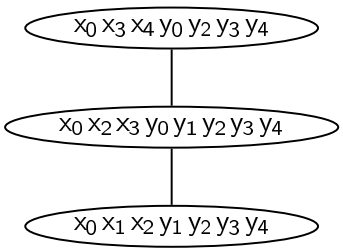

Figure 5: Junction tree for rule (2).

tree we derive from an SCFG rule has a cluster containing all four outer boundary variables, allowing efficient lookup of the inner maximization in (3). Because the outer boundary variables need not appear throughout the junction tree, this technique allows reuse of some partial results across different outer boundaries. As an example, consider message passing on the junction tree of shown in Figure 5, which corresponds to the parsing strategy of Figure 1. Only the final step involves all four boundaries of the complete cell, but the most complex step is the second, with a total of eight boundaries. This efficient reuse would not be achieved by applying the junction tree technique directly to the maximization operator in Algorithm 1, because we would be fixing the outer boundaries and computing the junction tree only over the inner boundaries.

3 Treewidth and Tabular Parsing

The complexity of the message passing algorithm over an MRF's junction tree is determined by the *treewidth* of the MRF. In this section we show that, because parsing strategies are in direct correspondence with valid junction trees, we can use treewidth to analyze the complexity of a grammar rule.

We define a tabular parsing strategy as any dynamic programming algorithm that stores partial results corresponding to subsets of a rule's child nonterminals. Such a strategy can be represented as a recursive partition of child nonterminals, as shown in Figure 1(left). We show below that a recursive partition of children having maximum complexity k at any step can be converted into a junction tree having k as the maximum cluster size. This implies that finding the optimal junction tree will give a parsing strategy at least as good as the strategy of the optimal recursive partition.

A recursive partition of child nonterminals can be converted into a junction tree as follows:

- For each leaf of the recursive partition, which represents a single child nonterminal i, create a leaf in the junction tree with the cluster $(x_{i-1}, x_i, y_{\pi(i)-1}, y_{\pi(i)})$ and the potential function $f_i(x_{i-1}, x_i, y_{\pi(i)-1}, y_{\pi(i)})$.
- For each internal node in the recursive partition, create a corresponding node in the junction tree.
- Add each variable x_i to all nodes in the junction tree on the path from the node for child nonterminal $i-1$ to the node for child nonterminal i. Similarly, add each variable $y_{\pi(i)}$ to all nodes in the junction tree on the path from the node for child nonterminal $\pi(i)-1$ to the node for child nonterminal $\pi(i)$.

Because each variable appears as an argument of only two factors, the junction tree nodes in which it is present form a linear path from one leaf of the tree to another. Since each variable is associated only with nodes on one path through the tree, the resulting tree will satisfy the junction tree property. The tree structure of the original recursive partition implies that the variable rises from two leaf nodes to the lowest common ancestor of both leaves, and is not contained in any higher nodes. Thus each node in the junction tree contains variables corresponding to the set of endpoints of the spans defined by the two subsets corresponding to its two children. The number of variables at each node in the junction tree is identical to the number of free endpoints at the corresponding combination in the recursive partition.

Because each recursive partition corresponds to a junction tree with the same complexity, finding the best recursive partition reduces to finding the junction tree with the best complexity, i.e., the smallest maximum cluster size.

Finding the junction tree with the smallest cluster size is equivalent to finding the input graph's *treewidth*, the smallest k such that the graph can be embedded in a k-tree. In general, this problem was shown to be NP-complete by Arnborg et al. (1987). However, because the treewidth of a given rule lower bounds the complexity of its tabular parsing strategies, parsing complexity for general rules can be

bounded with treewidth results for worst-case rules, without explicitly identifying the worst-case permutations.

4 Treewidth Grows Linearly

In this section, we show that the treewidth of the graphs corresponding to worst-case permutations growths linearly with the permutation's length. Our strategy is as follows:

1. Define a 3-regular graph for an input permutation consisting of a subset of edges from the original graph.
2. Show that the edge-expansion of the 3-regular graph grows linearly for randomly chosen permutations.
3. Use edge-expansion to bound the spectral gap.
4. Use spectral gap to bound treewidth.

For the first step, we define $H = (V, E)$ as a random 3-regular graph on $2n$ vertices obtained as follows. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be cycles, each on a separate set of n vertices. These two cycles correspond to the edges (x_i, x_{i+1}) and (y_i, y_{i+1}) in the graphs of the type shown in Figure 4. Let M be a random perfect matching between V_1 and V_2. The matching represents the edges $(x_i, y_{\pi(i)})$ produced from the input permutation π. Let H be the union of G_1, G_2, and M. While H contains only some of the edges in the graphs defined in the previous section, removing edges cannot increase the treewidth.

For the second step of the proof, we use a probabilistic argument detailed in the next subsection.

For the third step, we will use the following connection between the edge-expansion and the eigenvalue gap (Alon and Milman, 1985; Tanner, 1984).

Lemma 4.1 *Let G be a k-regular graph. Let λ_2 be the second largest eigenvalue of G. Let $h(G)$ be the edge-expansion of G. Then*

$$k - \lambda_2 \geq \frac{h(G)^2}{2k}.$$

Finally, for the fourth step, we use a relation between the eigenvalue gap and treewidth for regular graphs shown by Chandran and Subramanian (2003).

Lemma 4.2 *Let G be a k-regular graph. Let n be the number of vertices of G. Let λ_2 be the second largest eigenvalue of G. Then*

$$\mathrm{tw}(G) \geq \left\lfloor \frac{n}{4k}(k - \lambda_2) \right\rfloor - 1$$

Note that in our setting $k = 3$. In order to use Lemma 4.2 we will need to give a lower bound on the eigenvalue gap $k - \lambda_2$ of G.

4.1 Edge Expansion

The *edge-expansion* of a set of vertices T is the ratio of the number of edges connecting vertices in T to the rest of the graph, divided by the number of vertices in T,

$$\frac{|E(T, V - T)|}{|T|}$$

where we assume that $|T| \leq |V|/2$. The edge expansion of a graph is the minimum edge expansion of any subset of vertices:

$$h(G) = \min_{T \subseteq V} \frac{|E(T, V - T)|}{\min\{|T|, |V - T|\}}.$$

Intuitively, if all subsets of vertices are highly connected to the remainder of the graph, there is no way to decompose the graph into minimally interacting subgraphs, and thus no way to decompose the dynamic programming problem of parsing into smaller pieces.

Let $\binom{n}{k}$ be the standard binomial coefficient, and for $\alpha \in \mathbb{R}$, let

$$\binom{n}{\leq \alpha} = \sum_{k=0}^{\lfloor \alpha \rfloor} \binom{n}{k}.$$

We will use the following standard inequality valid for $0 \leq \alpha \leq n$:

$$\binom{n}{\leq \alpha} \leq \left(\frac{n\mathrm{e}}{\alpha}\right)^{\alpha} \tag{4}$$

Lemma 4.3 *With probability at least 0.98 the graph H has edge-expansion at least $1/50$.*

Proof :
Let $\varepsilon = 1/50$. Assume that $T \subseteq V$ is a set with a small edge-expansion, i. e.,

$$|E(T, V - T)| \leq \varepsilon |T|, \tag{5}$$

and $|T| \leq |V|/2 = n$. Let $T_i = T \cap V_i$ and let $t_i = |T_i|$, for $i = 1, 2$. We will w.l.o.g. assume $t_1 \leq t_2$. We will denote as ℓ_i the number of spans of consecutive vertices from E_i contained in T. Thus $2\ell_i = |E(T_i, V_i - T_i)|$, for $i = 1, 2$. The spans counted by ℓ_1 and ℓ_2 correspond to continuous spans counted in computing the complexity of a chart parsing operation. However, unlike in the diagrams in the earlier part of this paper, in our graph theoretic argument there is no requirement that T select only corresponding pairs of vertices from V_1 and V_2.

There are at least $2(\ell_1+\ell_2)+t_2-t_1$ edges between T and $V - T$. This is because there are $2\ell_i$ edges within V_i at the left and right boundaries of the ℓ_i spans, and at least $t_2 - t_1$ edges connecting the extra vertices from T_2 that have no matching vertex in T_1. Thus from assumption (5) we have

$$t_2 - t_1 \leq \varepsilon(t_1 + t_2)$$

which in turn implies

$$t_1 \leq t_2 \leq \frac{1+\varepsilon}{1-\varepsilon}\, t_1. \tag{6}$$

Similarly, using (6), we have

$$\ell_1 + \ell_2 \leq \frac{\varepsilon}{2}\,(t_1 + t_2) \leq \frac{\varepsilon}{1-\varepsilon}\, t_1. \tag{7}$$

That is, for T to have small edge expansion, the vertices in T_1 and T_2 must be collected into a small number of spans ℓ_1 and ℓ_2. This limit on the number of spans allows us to limit the number of ways of choosing T_1 and T_2. Suppose that t_1 is given. Any pair T_1, T_2 is determined by the edges in $E(T_1, V_1 - T_1)$, and $E(T_2, V_2 - T_2)$, and two bits (corresponding to the possible "swaps" of T_i with $V_i - T_i$). Note that we can choose at most $2\ell_1 + 2\ell_2 \leq t_1 \cdot 2\varepsilon/(1-\varepsilon)$ edges in total. Thus the number of choices of T_1 and T_2 is bounded above by

$$4 \cdot \binom{2n}{\leq \frac{2\varepsilon}{1-\varepsilon} t_1}. \tag{8}$$

For a given choice of T_1 and T_2, for T to have small edge expansion, there must also not be too many edges that connect T_1 to vertices in $V_2 - T_2$. Let k be the number of edges between T_1 and T_2. There are at least $t_1 + t_2 - 2k$ edges between T and $V - T$ and from assumption (5) we have

$$t_1 + t_2 - 2k \leq \varepsilon(t_1 + t_2)$$

Thus

$$k \geq (1-\varepsilon)\frac{t_1 + t_2}{2} \geq (1-\varepsilon)t_1. \tag{9}$$

The probability that there are $\geq (1-\varepsilon)t_1$ edges between T_1 and T_2 is bounded by

$$\binom{t_1}{\leq \varepsilon t_1} \left(\frac{t_2}{n}\right)^{(1-\varepsilon)t_1}$$

where the first term selects vertices in T_1 connected to T_2, and the second term upper bounds the probability that the selected vertices are indeed connected to T_2. Using 6, we obtain a bound in terms of t_1 alone:

$$\binom{t_1}{\leq \varepsilon t_1} \left(\frac{1+\varepsilon}{1-\varepsilon} \cdot \frac{t_1}{n}\right)^{(1-\varepsilon)t_1}, \tag{10}$$

Combining the number of ways of choosing T_1 and T_2 (8) with the bound on the probability that the edges M from the input permutation connect almost all the vertices in T_1 to vertices from T_2 (10), and using the union bound over values of t_1, we obtain that the probability p that there exists $T \subseteq V$ with edge-expansion less than ε is bounded by:

$$2 \sum_{t_1=0}^{\lfloor n/2 \rfloor} 4 \cdot \binom{2n}{\leq \frac{2\varepsilon}{1-\varepsilon} t_1} \binom{t_1}{\leq \varepsilon t_1} \left(\frac{1+\varepsilon}{1-\varepsilon} \cdot \frac{t_1}{n}\right)^{(1-\varepsilon)t_1} \tag{11}$$

where the factor of 2 is due to the assumption $t_1 \leq t_2$.

The graph H is connected and hence T has at least one out-going edge. Therefore if $t_1 + t_2 \leq 1/\varepsilon$, the edge-expansion of T is at least ε. Thus a set with edge-expansion less than ε must have $t_1 + t_2 \geq 1/\varepsilon$, which, by (6), implies $t_1 \geq (1-\varepsilon)/(2\varepsilon)$. Thus the sum in (11) can be taken for t from $\lceil (1-\varepsilon)/(2\varepsilon) \rceil$

to $\lfloor n/2 \rfloor$. Using (4) we obtain

$$p \leq 8 \sum_{t_1=\lceil \frac{1-\varepsilon}{2\varepsilon} \rceil}^{\lfloor n/2 \rfloor} \left[\left(\frac{2n\mathrm{e}}{\frac{2\varepsilon}{1-\varepsilon} t_1} \right)^{\frac{2\varepsilon}{1-\varepsilon} t_1} \left(\frac{t_1 \mathrm{e}}{\varepsilon t_1} \right)^{\varepsilon t_1} \left(\frac{1+\varepsilon}{1-\varepsilon} \cdot \frac{t_1}{n} \right)^{(1-\varepsilon)t_1} \right] =$$

$$8 \sum_{t_1=\lceil \frac{1-\varepsilon}{2\varepsilon} \rceil}^{\lfloor n/2 \rfloor} \left(\left(\frac{\mathrm{e}(1-\varepsilon)}{\varepsilon} \right)^{\frac{2\varepsilon}{1-\varepsilon}} \left(\frac{\mathrm{e}}{\varepsilon} \right)^{\varepsilon} \left(\frac{1+\varepsilon}{1-\varepsilon} \right)^{1-\varepsilon} \left(\frac{t_1}{n} \right)^{1-\varepsilon-\frac{2\varepsilon}{1-\varepsilon}} \right)^{t_1}. \quad (12)$$

We will use $t_1/n \leq 1/2$ and plug $\varepsilon = 1/50$ into (12). We obtain

$$p \leq 8 \sum_{t_1=25}^{\infty} 0.74^{t_1} \leq 0.02.$$

■

While this constant bound on p is sufficient for our main complexity result, it can further be shown that p approaches zero as n increases, from the fact that the geometric sum in (12) converges, and each term for fixed t_1 goes to zero as n grows.

This completes the second step of the proof as outlined at the beginning of this section. The constant bound on the edge expansion implies a constant bound on the eigenvalue gap (Lemma 4.1), which in turn implies an $\Omega(n)$ bound on treewidth (Lemma 4.2), yielding:

Theorem 4.4 *Tabular parsing strategies for Synchronous Context-Free Grammars containing rules with all permutations of length n require time $\Omega(N^{cn})$ in the input string length N for some constant c.*

We have shown our result without explicitly constructing a difficult permutation, but we close with one example. The zero-based permutations of length p, where p is prime, $\pi(i) = i^{-1} \mod p$ for $0 < i < p$, and $\pi(0) = 0$, provide a known family of expander graphs (see Hoory et al. (2006)).

5 Conclusion

We have shown in the exponent in the complexity of polynomial-time parsing algorithms for synchronous context-free grammars grows linearly with the length of the grammar rules. While it is very expensive computationally to test whether a specified permutation has a parsing algorithm of a certain complexity, it turns out that randomly chosen permutations are difficult with high probability.

Acknowledgments This work was supported by NSF grants IIS-0546554, IIS-0428020, and IIS-0325646.

References

Albert V. Aho and Jeffery D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling*, volume 1. Prentice-Hall, Englewood Cliffs, NJ.

N. Alon and V.D. Milman. 1985. λ_1, isoperimetric inequalities for graphs and superconcentrators. *J. of Combinatorial Theory, Ser. B*, 38:73–88.

Stefen Arnborg, Derek G. Corneil, and Andrzej Proskurowski. 1987. Complexity of finding embeddings in a k-tree. *SIAM Journal of Algebraic and Discrete Methods*, 8:277–284, April.

L.S. Chandran and C.R. Subramanian. 2003. A spectral lower bound for the treewidth of a graph and its consequences. *Information Processing Letters*, 87:195–200.

Shlomo Hoory, Nathan Linial, and Avi Wigderson. 2006. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, 43:439–561.

Finn V. Jensen, Steffen L. Lauritzen, and Kristian G. Olesen. 1990. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4:269–282.

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

G. Shafer and P. Shenoy. 1990. Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2:327–353.

R.M. Tanner. 1984. Explicit construction of concentrators from generalized n-gons. *J. Algebraic Discrete Methods*, 5:287–294.