Finding the Smallest Binarization of a CFG is NP-Hard

Carlos Gómez-Rodríguez[*]

Departamento de Computación, Universidade da Coruña, Spain

Abstract

Grammar binarization is the process and result of transforming a grammar to an equivalent form whose rules contain at most two symbols in their right-hand side. Binarization is used, explicitly or implicity, by a wide range of parsers for context-free grammars and other grammatical formalisms. Non-trivial grammars can be binarized in multiple ways, but in order to optimize the parser's computational cost, it is convenient to choose a binarization that is as small as possible. While several authors have explored heuristics to obtain compact binarizations, none of them guarantee that the resulting grammar has minimum size. However, to our knowledge, no hardness results for this problem have been published. In this article, we address this issue and prove that the problem of finding a minimum binarization of a given context-free grammar is NP-hard, by reduction from vertex cover. We also provide a lower bound on the approximability of this problem.

Keywords: natural language processing, complexity, binarization, grammars, context-free grammar, parsing
2000 MSC: 68 Computer Science

NOTICE: this is the author's version of a work that was accepted for publication in Journal of Computer and System Sciences. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was submitted for publication. The definitive version will appear in
`http://www.sciencedirect.com/science/article/pii/S0022000013002055`

[*]Corresponding address: Facultade de Informática, Campus de Elviña, s/n, 15071 A Coruña, Spain. Tel: +34 981 167000 ext. 1377, Fax: +34 981 167160.

Email address: `cgomezr@udc.es` (Carlos Gómez-Rodríguez)

URL: `http://www.grupolys.org/~cgomezr` (Carlos Gómez-Rodríguez)

1. Introduction

The process of grammar binarization, by which a grammar is transformed into an equivalent binary form by splitting its larger rules into sets of binary rules, is essential to perform efficient natural language parsing with context-free grammars (CFGs) and other grammatical formalisms.

The most widely known and discussed application of binarization is probably the preprocessing step for the CKY parsing algorithm for CFGs [1, 2], which needs to use a binary grammar in order to achieve $O(n^3)$ time complexity.[1] However, most if not all the well-known CFG parsing algorithms use binarization in one way or another: although algorithms such as the Earley [5], Left-Corner [6], Graham-Harrison-Ruzzo [7] or Head-Corner [8] parsers do not perform it explicitly as a preprocessing step, they binarize rules implicitly using dotted items. In fact, some of these parsers (such as the Left-Corner [6] and Head-Corner [8] parsers) can be seen as implementations of the same underlying parser with different binarization strategies.

Binarization is also used to achieve efficient parsing for more powerful grammatical formalisms, such as conjunctive grammars [9], synchronous context-free grammars [10], linear context-free rewriting systems [11], coupled context-free grammars [12] or regular tree grammars [13]. While in this article we will only deal with CFGs, the hardness result presented in this article is applicable to all these formalisms as they contain CFGs as a particular case. Note that, in the case of synchronous CFGs and linear context-free rewriting systems with bounded fan-out, this is arguably not very useful because these formalisms do not guarantee that a binarization of any size even *exists* for a given grammar [10, 14]. However, it can still be applied to subclasses of these formalisms that do guarantee this property, such as well-nested linear context-free rewriting systems [15] or binarizable synchronous CFGs [10].

A CFG can be binarized in different ways, depending on which pairs of symbols are chosen to be grouped to form binary rules. In particular, the number of possible binarizations for a single $(n+1)$-ary CFG production rule is the nth Catalan number, $C_n = \frac{1}{n+1}\binom{2n}{n}$ [16]. This naturally raises the

[1]While CKY is traditionally described for grammars in Chomsky Normal Form [3], which impose some extra conditions apart from being binary (e.g. disallowing rules with a single nonterminal in the right-hand side), it is straightforward to define it in a more general way that can work efficiently with any binary grammar [4].

question of which binarization should be used to obtain the best parsing efficiency. Since grammar size is an important factor in the efficiency of natural language parsers [17], an obvious approach is to binarize in a way that tries to collapse pairs of nonterminal symbols that are frequent in grammar rules, so as to obtain a binary grammar as compact as possible [18]. Song et al. [16] provide empirical evidence that using compact grammars greatly improves the efficiency of a CKY parser with respect to naive binarization, and gain further improvements by using corpus statistics to predict the amount of useless items (items not leading to a complete parse) that will be generated by different binarizations.

While several authors have recently tackled the problem of searching for small binarizations for a given grammar [18, 16, 19]; no efficient algorithm has been described that will solve the problem of finding the smallest size binarization for a given CFG (the *minimum binarization problem*). Instead, the mentioned papers use greedy algorithms to find a reasonably small binarization, without guarantee of optimality. An experiment by DeNero et al. [19] for transducer grammars showed this strategy to output a grammar more than seven times smaller than naive right-branching binarization, but still more than twice the size of the optimal grammar obtained using an integer linear programming solver (which, reportedly, could only find optimal solutions for very small grammars).

To the best of our knowledge, in spite of the absence of an efficient algorithm to compute the optimal binarization for a given CFG; and even though growing use of mildly context-sensitive formalisms such as synchronous context-free grammars, linear context-free rewriting systems or range concatenation grammars has led to a surge of interest in theoretical results about binarization in the last few years [11, 10, 15, 20]; no hardness results are known for the minimum binarization problem.

In this article, we fill that gap in the literature by proving that the minimum binarization problem is NP-hard. We do so by reduction from the vertex cover problem, inspired by the proof for the (related, but not equivalent) smallest grammar problem by Charikar et al. [21]. We also provide an inapproximability result, showing that it is NP-hard to approximate the minimum binarization problem within any factor smaller than 2575/2574.

Related work. As already mentioned, the problem of finding a binarization as small as possible for a given grammar has been considered in several recent papers in the computational linguistics literature [18, 16, 19]; but we know of no published hardness results for the minimum binarization problem.

On the other hand, the related problem of finding the minimum (not necessarily binary) context-free grammar that defines a given single-string language, which is relevant in data compression [22], is known to be NP-hard. While this problem (called the "smallest grammar problem" [21]) is not equivalent to the minimum binarization problem, they do have obvious similarities, and the hardness proof for the minimum binarization problem in Section 3 has been inspired by the proof for the smallest grammar problem given by Charikar et al. [21].

In particular, the main differences between the two problems are the following:

1. The smallest grammar problem is concerned with a language with a single string (or, equivalently, it starts from a grammar with a single rule), whereas the minimum binarization problem can have any context-free grammar as input.
2. In relation to the previous point, the smallest grammar problem is not limited to transformations that preserve the structure of parse trees: it only requires so-called *weak equivalence*, i.e., that the input and output grammars generate the same set of strings. On the other hand, as will be explained in detail in Section 2, the minimum binarization problem requires that the output grammar be a factorization of the input grammar, i.e., a transformation of the input obtained by splitting its rules into smaller rules. This is so that the parse trees under the original grammar can be recovered from parses obtained with the binarized one, and the string and tree probabilities under the original grammar carry over to the binarized one if a probabilistic model is used.
3. In the minimum binarization problem, size is minimized among binary grammars only, while in the smallest grammar problem any context-free grammar is allowed. Note that this difference is relevant because the smallest binary CFG for a given string or language may be different to (and larger than) the smallest unrestricted CFG for that string: for example, we can generate the language $\{abc\}$ with a non-binary grammar of size 3 (the one with a single rule $S \rightarrow abc$, with three symbols on its right-hand side) but the smallest binary grammars for that same language have size 4 (one of them would have the rules $S \rightarrow Ac, A \rightarrow ab$, for a total of four symbols in their right-hand sides).

While the first two differences would not prevent us from carrying over the core of the hardness proof from the smallest grammar problem to the minimum binarization problem, the third one does, since the proof by Charikar et al. [21] relies on the construction of a grammar whose optimized version has non-binary rules. It is also not obvious how to attempt a proof by reduction of the smallest grammar problem to the minimum binarization problem, since we do not know whether the smallest binary grammar need always be a binarization of the smallest grammar, and even supposing that this proposition holds, we would still need to find a polynomial way of finding the optimal set of pairs of nonterminals to collapse (or of pairs of binary rules to merge into larger rules) to obtain the smallest grammar from the smallest binarization.

Thus, rather than attempting such a reduction, we will prove that the minimum binarization problem is NP-hard by using a new proof by reduction from vertex cover, inspired on the one in [21] (which was, in turn, inspired by arguments given by Storer and Szymanski [23]) but using a different grammar construction at its core to adapt it to our problem. As we will see, this new grammar construction produces a somewhat simpler proof than the one for the smallest grammar problem by Charikar et al. [21].

2. The Minimum Binarization Problem

In this section, we formally define the problem of finding the minimum-size binarization of a CFG, introduced informally in Section 1. To do so, we first introduce some preliminary concepts, starting with the definition of a context-free grammar.

Definition 1. *A* context-free grammar (CFG) *is a tuple* $G = (N, \Sigma, P, S)$, *such that:*

- Σ *is a finite alphabet of* terminal *symbols,*
- N *is a finite set of* nonterminal *symbols such that* $\Sigma \cap N = \emptyset$,
- $S \in N$ *is the* start symbol *of the grammar, and*
- P *is a finite set of* production rules *of the form* $A \to \alpha$, *where* $A \in N$ *is a non-terminal symbol, and* $\alpha \in (N \cup \Sigma)^\star$ *is a string over terminal and nonterminal symbols. We say that* A *is the* left-hand side *of the rule* $A \to \alpha$, *and that* α *is its* right-hand side.

Note that, following standard notational conventions in the literature [24], we will use lowercase letters $a, b, c \dots$ to denote terminal symbols, uppercase letters $A, B, C \dots$ for nonterminal symbols, uppercase letters $X, Y, Z \dots$ for symbols in $N \cup \Sigma$, and lowercase Greek letters $(\alpha, \beta, \gamma, \dots)$ for strings over terminal and nonterminal symbols. The letter ϵ will be used to denote the empty string.

Given a CFG $G = (N, \Sigma, P, S)$, we define the *derivation* relation $\Rightarrow \subseteq (N \cup \Sigma)^\star \times (N \cup \Sigma)^\star$ as follows: we say that a string $\alpha A \beta$ derives a string $\alpha\gamma\beta$ ($\alpha A \beta \Rightarrow \alpha\gamma\beta$) if, and only if, there is a rule $A \to \gamma \in P$. The *string language* generated by the grammar G, written $L(G)$, is given by $L(G) = \{\alpha \in \Sigma^\star \mid S \Rightarrow^\star \alpha\}$, where $\Rightarrow^\star$ is the reflexive-transitive closure of the derivation relation. Two grammars are said to be *weakly equivalent* if they generate the same string language.

However, a context-free grammar is not only used to define a string language, but also to associate a set of hierarchical structures (called *parse trees*) to each string in the language. A finite ordered tree is a *parse tree* over the grammar $G = (N, \Sigma, P, S)$ if it satisfies the following conditions:

- The root node is labelled S,
- Leaf nodes have a label in $\Sigma \cup \{\epsilon\}$,
- Non-leaf nodes have a label in N,
- If a non-leaf node labelled A has children $X_1, \dots, X_n$ ($X_i \in N \cup \Sigma \cup \{\epsilon\}$), then $A \to X_1 \dots X_n \in P$.

The *tree language* generated by the grammar G, written $L_T(G)$, is the set of all parse trees over G. The *yield* of a parse tree τ, written $\mathit{yield}(\tau)$, is the string that is obtained by concatenating the labels of all its leaf nodes in left-to-right order, and we then say that τ is a parse tree for the string $\mathit{yield}(\tau)$. It is easy to check that

$$L(G) = \{\mathit{yield}(\tau) \mid \tau \in L_T(G)\}$$

i.e., the string language of a grammar is the set of yields of its tree language (or equivalently, the set of strings that have a parse tree over G).

In this paper, we are interested in obtaining grammars that are binary, i.e., have at most two symbols in each of their rules' right-hand side:

Definition 2. *A CFG $G = (N, \Sigma, P, S)$ is said to be* binary *if $|\alpha| \leq 2$ for all $A \to \alpha \in P$.*

The goal of grammar binarization, as described in Section 1, is to transform a non-binary CFG G into an equivalent CFG G' that is binary. Since in natural language parsing we are not only interested in knowing whether sentences belong to the string language $L(G)$ or not, but also on obtaining their parse trees (and, in probabilistic parsing, their associated probabilities); the notion that G and G' must be *equivalent* not only means that they must generate the same string language (weak equivalence), but also that the set of parse trees generated by the original grammar G can be obtained from the set of parse trees generated by the binarized grammar G' by means of a homomorphism. In the CFG literature, this corresponds to the notion that G is *covered* by G' [25, 26].

This requirement means that, when searching for binarizations, we are limited to grammars that are obtained from G by applying *factorization* [20], i.e., by splitting production rules into sets of smaller rules, each containing a smaller number of variables than the original rule:

Definition 3. *Let $\Pi = A \to X_1 \dots X_n$ be a context-free rule. The $X_i \dots X_j$-*reduction *of Π, written $\rho_{X_i \dots X_j}(\Pi)$, is a set formed by the following two rules:*

$$\rho^1_{X_i \dots X_j}(\Pi) = A \to X_1 \dots X_{i-1}[X_i \dots X_j]X_{j+1} \dots X_n$$

$$\rho^2_{X_i \dots X_j}(\Pi) = [X_i \dots X_j] \to X_i \dots X_j$$

where $[X_i \dots X_j]$ is a fresh nonterminal symbol.

This means that an $X_i \dots X_j$-reduction of Π splits the original production Π into two new productions, $\rho^1_{X_i \dots X_j}(\Pi)$ and $\rho^2_{X_i \dots X_j}(\Pi)$, by collapsing a sequence of symbols in the rule's right-hand side into a fresh nonterminal $[X_i \dots X_j]$. The length of the right-hand side of each of the productions resulting from a reduction is always strictly smaller than the length of the right-hand side of the original rule. It is important to note that if one of the nonterminals $X_i \dots X_j$ was in turn a fresh nonterminal symbol created by another reduction, we will remove inner brackets, e.g., the nonterminal created by an $A[BC]$-reduction will be written as $[ABC]$, and is the same symbol as the nonterminal created by a $[AB]C$-reduction.

Example 1. *The AB-reduction of the context-free rule $S \to ABC$ is the set containing the following two rules:*

$$\rho^1_{AB}(S \to ABC) = S \to [AB]C,$$
$$\rho^2_{AB}(S \to ABC) = [AB] \to AB.$$

If instead of an AB-reduction we apply a BC-reduction, we will obtain the following rules instead:

$$\rho^1_{BC}(S \to ABC) = S \to A[BC],$$
$$\rho^2_{BC}(S \to ABC) = [BC] \to BC.$$

In this case, both reductions split the rule $S \to ABC$, which has three symbols in its right-hand side, into two binary rules.

Note that a reduction transformation can easily be undone, obtaining the original production rule, and that it can trivially be adapted so that it preserves parse tree probabilities in probabilistic extensions of context-free grammars [16].

Now that we have defined the concept of reduction for a context-free rule, we can extend it to grammars:

Definition 4. *Let $G = (N, \Sigma, P, S)$ be a CFG. We say that a CFG $G' = (N', \Sigma, P', S)$ is a* direct reduction *of G (written $G \rhd G'$) if there exists a production rule $\Pi = A \to X_1 \dots X_n \in P$ and symbols $X_i \dots X_j$ such that:*

- $N' = N \cup \{[X_i \dots X_j]\}$,
- $P' = (P \setminus \{\Pi\}) \cup \rho_{X_i \dots X_j}(\Pi)$,

i.e., if G' is obtained from G by reducing one of its rules.

We can now define a binarization of G as a binary grammar obtained by applying zero or more reductions to G:

Definition 5. *Let $G = (N, \Sigma, P, S)$ be a CFG. We say that a CFG G' is a* binarization *of G if G' is binary and $G \rhd^\star G'$, where $\rhd^\star$ is the reflexive-transitive closure of the direct reduction relation. We will write $Bin(G)$ for the set of all binarizations of G.*

A minimum binarization of a CFG will be a binarization that minimizes grammar size. We define the size of a CFG G, denoted by $|G|$, as the total number of symbols in all of its right-hand sides:

$$|G| = \sum_{A \to \alpha \in P} |\alpha|$$

.

With this, we are ready to define the *minimum binarization(s)* of G:

Definition 6. *Let $G = (N, \Sigma, P, S)$ be a CFG. The set of* minimum binarizations *of G, written $mb(G)$, is*

$$mb(G) = \operatorname*{argmin}_{G' \in Bin(G)} |G'|$$

Example 2. *Let $G = (N, \Sigma, P, S)$ be a grammar such that*

$$\begin{aligned} P = \{S &\to abC, \\ C &\to abD, \\ D &\to d\}. \end{aligned}$$

This grammar generates the string language $\{ababd\}$, and its tree language contains a single tree $S(abC(abD(d)))$.

By applying a bC-reduction to $S \to abC$ and a bD-reduction to $C \to abD$, we obtain a binarization G'_1 with the following rules:

$$\begin{aligned} P'_1 = \{S &\to a[bC], \\ C &\to a[bD], \\ D &\to d, \\ [bC] &\to bC, \\ [bD] &\to bD\}, \end{aligned}$$

which has size 9. If instead of these reductions we apply an ab-reduction to both $S \to abC$ and $C \to abD$, we obtain a binarization G'_2 of size 7:

$$\begin{aligned} P'_2 = \{S &\to [ab]C, \\ C &\to [ab]D, \\ D &\to d, \\ [ab] &\to ab\}. \end{aligned}$$

In this toy example, it is easy to check that G'_2 is the smallest possible binarization of G, and thus the unique member of $mb(G)$.

Note that it is possible to obtain a smaller binary grammar that generates the same string language by removing the production $D \rightarrow d$ and using the terminal symbol d directly in the rule $C \rightarrow [ab]D$, thus changing it to $C \rightarrow [ab]d$. However, the resulting grammar (of length 6) is not a binarization, as it cannot be obtained by applying reduction operations and, as a consequence, it is not structurally equivalent to the grammar G (we cannot recover the parse trees with respect to G from parse trees obtained with this grammar, since it no longer contains any rule reflecting the fact that a word d is a daughter of D – in contrast, any binarization, such as G'_2, contains all the information necessary to reconstruct the parse trees of G).

The minimum binarization problem consists on finding one member of $mb(G)$ given a grammar G. We are now ready to prove that this problem is NP-hard.[2]

3. Hardness

We will now prove the main result of this article: that the minimum binarization problem is NP-hard.

To do so, we show how the well-known minimum vertex cover problem [28] can be reduced to the minimum binarization problem.

Given an undirected graph $\mathcal{G} = (V, E)$, a *vertex cover* for G is a subset $V' \subseteq V$ such that, for each edge $(u, v) \in E$, at least one of the nodes u, v is in V'. The *minimum vertex cover* problem is the optimization problem of finding a vertex cover of G with minimum cardinality, and it is known to be NP-hard [28].

Note that, for simplicity, we will assume that all the graphs we are working with are undirected. This is not relevant for the validity of the proof, which

[2]Note that the problem of finding the smallest binary grammar that is weakly equivalent to a given CFG G (i.e., the smallest binary grammar that generates $L(G)$, without necessarily being a binarization of G) is known to be undecidable. This can be seen by considering that an algorithm that solved this problem could be used to solve the CFG universality problem, which is undecidable [27]. In contrast, the minimum binarization problem is trivially decidable, as the number of possible binarizations for a given grammar is finite.

also stands if directed graphs (and the associated variant of the vertex cover problem) are used instead.

Let $\mathcal{G} = (V, E)$ be a graph. To reduce the minimum vertex cover problem to the minimum binarization problem, we define a context-free grammar $G(\mathcal{G}) = (\{S\}, \Sigma(\mathcal{G}), P(\mathcal{G}), S))$, where S is a nonterminal that we use as the start symbol of $G(\mathcal{G})$; the terminal alphabet $\Sigma(\mathcal{G})$ is defined as follows:

$$\Sigma(\mathcal{G}) = V \cup \{x_v^i \mid v \in V, i \in \{1, 2, 3, 4\}\} \cup \{\$\},$$

meaning that each graph node $v \in V$ acts as a terminal symbol, and we also define four additional symbols $x_v^1, \ldots, x_v^4$ per graph node, and an additional padder symbol \$; and the set of productions $P(\mathcal{G})$ is defined as follows:

$$P(\mathcal{G}) = P_1(\mathcal{G}) \cup P_2(\mathcal{G}),$$

with

$$\begin{aligned} P_1(\mathcal{G}) = \{&S \to v\$x_v^1 \mid v \in V\} \\ \cup\{&S \to v\$x_v^2 \mid v \in V\} \\ \cup\{&S \to x_v^3\$v \mid v \in V\} \\ \cup\{&S \to x_v^4\$v \mid v \in V\}, \end{aligned}$$

and

$$P_2(\mathcal{G}) = \{S \to \$u\$v\$ \mid (u, v) \in E\}.$$

Note that the set of productions $P(\mathcal{G})$ contains $|E| + 4|V|$ production rules: four productions per graph node, plus one additional production per edge.

The following two examples illustrate the vertex cover problem for a particular example graph $\mathcal{G}_e$, and show the associated grammar $G(\mathcal{G}_e)$ obtained from it:

Example 3. *Consider the graph $\mathcal{G}_e = (V_e, E_e)$ with $V_e = \{a, b, c, d, e\}$ and $E_e = \{(a, b), (a, c), (b, d), (c, d), (d, e)\}$, depicted in Figure 1. The set $\{b, c, e\}$ is a vertex cover of $\mathcal{G}_e$ with cardinality 3. The set $\{a, d\}$, of cardinality 2, is the minimum vertex cover of $\mathcal{G}_e$. Note that a graph may have more than one minimum vertex cover (for example, if we removed the edge (d, e) from $\mathcal{G}_e$, then both $\{a, d\}$ and $\{b, c\}$ would be minimum vertex covers).*

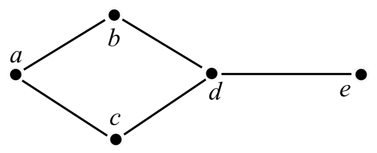

Figure 1: The graph $\mathcal{G}_e$ in Example 3, whose minimum vertex cover is the set of vertices $\{a, d\}$.

Example 4. *The grammar $G(\mathcal{G}_e)$ corresponding to the graph $\mathcal{G}_e$ of Example 3 has a terminal alphabet*

$$\Sigma(\mathcal{G}_e) = \{a, b, c, d, e\} \cup \{x_a^1, \ldots, x_a^4, \ldots, x_e^1, \ldots, x_e^4\} \cup \{\$\},$$

and its productions are given by

$$\begin{aligned} P_1(\mathcal{G}_e) = \{&S \to a\$x_a^1, \ldots, S \to e\$x_e^1, \\ &S \to a\$x_a^2, \ldots, S \to e\$x_e^2, \\ &S \to x_a^3\$a, \ldots, S \to x_e^3\$e, \\ &S \to x_a^4\$a, \ldots, S \to x_e^4\$e\}, \end{aligned}$$

$$\begin{aligned} P_2(\mathcal{G}_e) = \{&S \to \$a\$b\$,\ S \to \$a\$c\$,\ S \to \$b\$d\$, \\ &S \to \$c\$d\$,\ S \to \$d\$e\$\}. \end{aligned}$$

We now define a mapping from binarizations of the grammar $G(\mathcal{G})$ to sets of vertices of the graph G as follows:

Definition 7. *Let $G' = (N', \Sigma', P', S') \in Bin(G(\mathcal{G}))$ be a binarization of $G(\mathcal{G})$. Its* corresponding vertex set*, written $\mathcal{M}(G')$, is*

$$\mathcal{M}(G') = \{v \in V \mid X \Rightarrow^\star \$v\$ \text{ for some } X \in N'\}.$$

This means that we associate with each binarization G' the vertex set $\mathcal{M}(G')$ that contains all vertices v of $\mathcal{G}$ such that there is some nonterminal symbol that expands to $\$v\$$ in G'.

To obtain the desired reduction, we will now show that the vertex set for a minimum binarization of $G(\mathcal{G})$ provides us with a minimum vertex cover of the graph $\mathcal{G}$:

Lemma 1. *Let $\mathcal{G} = (V, E)$ be a graph, and $G' = (N', \Sigma', P', S') \in mb(G(\mathcal{G}))$ be a minimum binarization of its associated grammar $G(\mathcal{G})$. Then, the corresponding vertex set of G', $\mathcal{M}(G')$, is a minimum vertex cover of $\mathcal{G}$.*

Proof. We start by observing the following property of minimum binarizations of $G(\mathcal{G})$: any such binarization $G' = (N', \Sigma', P', S') \in mb(G(\mathcal{G}))$ contains the productions $[v\$] \to v\$$ and $[\$v] \to \$v$ for each $v \in V$.

To see why this holds, consider the productions in $P_1(\mathcal{G})$. Each of these productions has three symbols in its right-hand side, so in order to transform it into a binary production we need to apply a single reduction operation.

In particular, given a vertex $v \in V$, for the production in $P_1(\mathcal{G})$ of the form $S \to v\$x_v^1$ we have two choices: we can apply either a $v\$$-reduction or a $\$x_v^1$-reduction; and for the production of the form $S \to v\$x_v^2$ we can apply either a $v\$$-reduction or a $\$x_v^2$-reduction. It is easy to see that applying a $v\$$-reduction to both productions will always produce a smaller binarization than the other combinations, since the two ternary productions are reduced to three binary productions

$$[v\$] \to v\$, \;\; S \to [v\$]x_v^1, \;\; S \to [v\$]x_v^2$$

for a total size of 6, while any other combination of reductions will produce a larger set of binary productions due to having to write two different binary productions for each of the original productions, rather than sharing the nonterminal $[v\$]$ between both in order to share the production $[v\$] \to v\$$. It is important to note that, while we are reasoning only on a subset of the grammar, this locally optimal decision is also globally optimal: since by construction the substrings $\$x_v^1$ and $\$x_v^2$ do not appear in any other productions in the grammar $G(\mathcal{G})$, the productions and nonterminals produced by choosing a $\$x_v^1$-reduction or a $\$x_v^2$-reduction cannot be shared or reused to binarize other grammar rules, so choosing such reductions can never be beneficial.

By applying symmetric reasoning to the productions of the form $S \to x_v^3\$v$ and $S \to x_v^4\$v$ in $P_1(\mathcal{G})$, we show that a minimum binarization must always apply a $\$v$-reduction and thus contain the productions

$$[\$v] \to \$v, \;\; S \to x_v^3[\$v], \;\; S \to x_v^4[\$v].$$

This establishes the property that any minimum binarization of $G(\mathcal{G})$ contains the productions $[v\$] \to v\$$ and $[\$v] \to \$v$ for each $v \in V$.

Knowing this, we can now turn our attention to the productions in $P_2(\mathcal{G})$. For each edge (u, v) in $\mathcal{G}$, we have a production of the form $\Pi_{uv} = S \to \$u\$v\$$, with five symbols in the right-hand side. By the above property, we know that any minimum binarization of $G(\mathcal{G})$ must necessarily contain the productions

$$[\$u] \to \$u, \quad [u\$] \to u\$, \quad [\$v] \to \$v, \quad [v\$] \to v\$$$

which we can take advantage of when binarizing the production. Taking this into account, there are two locally optimal ways of binarizing Π_{uv}:

1. Applying a $[\$u]\$$-reduction (or, equivalently, a $\$[u\$]$-reduction – but we asume the former without loss of generality) after $[\$u]$- and $[v\$]$-reductions, which uses (and therefore, needs to add to the binarization) the productions

$$\begin{aligned}\Pi_1^u &= [\$u\$] \to [\$u]\$,\\ \Pi_2^u(u,v) &= S \to [\$u\$][v\$]\end{aligned}$$

   in addition to the productions with $[\$u]$ and $[v\$]$ in the left-hand side, which already had to be present by the property proved above.
2. Applying a $[\$v]\$$-reduction (or, equivalently, a $\$[v\$]$-reduction – but we asume the former without loss of generality) after $[\$u]$- and $[\$v]$-reductions, which uses the productions

$$\begin{aligned}\Pi_1^v &= [\$v\$] \to [\$v]\$,\\ \Pi_2^v(u,v) &= S \to [\$u][\$v\$]\end{aligned}$$

   in addition to the already-present productions for $[\$v]$ and $[\$u]$.

These two alternatives add 4 symbols to the size of the binarization, and it is easy to check case by case that any combination of reductions not falling into these two cases (e.g. those that use nonterminals of the form $[u\$v]$ or $[\$u\$v]$) create more rules and always produce binarizations that are not minimal, since a smaller one can always be found by using the above two choices of reductions. Therefore, any minimum binarization will contain either Π_1^u and $\Pi_2^u(u,v)$, or Π_1^v and $\Pi_2^v(u,v)$, for each edge (u,v) of $\mathcal{G}$.

Putting this together with the previous observations about the productions in $P_1(\mathcal{G})$, we know that any minimum binarization G' of $G(\mathcal{G})$ contains the following productions:

1. $|V|$ productions of the form $[v\$] \to v\$$, one for each $v \in V$.
2. $|V|$ productions of the form $[\$v] \to \$v$, one for each $v \in V$.

3. $4|V|$ productions of the form $S \to [v\$]x_v^1$, $S \to [v\$]x_v^2$, $S \to x_v^3[\$v]$, $S \to x_v^4[\$v]$, four for each $v \in V$.
4. $|E|$ productions of the forms $\Pi_2^u(u,v) = S \to [\$u\$][v\$]$ and $\Pi_2^v(u,v) = S \to [\$u][\$v\$]$; since for each edge $(u,v) \in E$, we have either $\Pi_2^u(u,v)$ or $\Pi_2^v(u,v)$, but not both.
5. $|\mathcal{M}(G')|$ productions of the form Π_1^v. The reason that the number of productions of this form is $|\mathcal{M}(G')|$ is that, by definition, the subset of vertices $v \in V$ such that there is a production of the form Π_1^v in G' is exactly $\mathcal{M}(G')$, the corresponding vertex set of G' (see Definition 7).

We will call any binarization of $G(\mathcal{G})$ that consists of productions of this form a *sensible* binarization of $G(\mathcal{G})$. Note that, with the results that we have shown up to this point, we know that every minimum binarization of $G(\mathcal{G})$ is sensible, but a sensible binarization of $G(\mathcal{G})$ need not be minimum. In particular, the size of a sensible binarization G', which is $2(6|V| + |E| + |\mathcal{M}(G')|)$, depends on the number of productions of the fifth form that it contains ($|\mathcal{M}(G')|$), and therefore the set of minimum binarizations is the set of sensible binarizations that minimize $|\mathcal{M}(G')|$.

We can now show that, given a sensible binarization G' of $G(\mathcal{G})$, the set $\mathcal{M}(G')$ is a vertex cover of the graph $\mathcal{G}$, since for each edge (u,v) in $\mathcal{G}$, either the production rule $\Pi_2^u(u,v)$ is in the binarization implying that Π_1^u is in the binarization too (and then $u \in \mathcal{M}(G')$) or $\Pi_2^v(u,v)$ is in the binarization, together with Π_1^v (and then $v \in \mathcal{M}(G')$). Conversely, it is also easy to show that every vertex cover of $\mathcal{G}$ corresponds to at least one sensible binarization: such a binarization can be obtained by including a production Π_1^v for each element v in the vertex cover, and choosing between the production $\Pi_2^u(u,v)$ and $\Pi_2^v(u,v)$ for each edge (u,v) according to which of the nodes u, v belongs to the vertex cover (if both nodes are in the vertex cover, any of these two productions can be chosen indistinctly).

This means that the mapping from sensible binarizations G' to vertex covers $\mathcal{M}(G')$ is surjective. Therefore, minimum binarizations, which are the sensible binarizations that minimize $|\mathcal{M}(G')|$, correspond to those where $|\mathcal{M}(G')|$ is a *minimum* vertex cover of $\mathcal{G}$, which proves Lemma 1. Note that the number of productions in each minimum binarization of $G(\mathcal{G})$ is then $6|V| + |E|$ plus the size of the minimum vertex covers of $\mathcal{G}$; and the size of minimum binarizations of $G(\mathcal{G})$ is twice that amount, since each of their productions has exactly two symbols on its right-hand side. □

Once we have established this lemma, the rest of the hardness proof is trivial: to solve an instance of the vertex cover problem for a graph $\mathcal{G} = (V, E)$, we build its associated grammar $G(\mathcal{G})$, we obtain a minimum binarization of it, and then we recover its corresponding vertex set by checking which terminals of the form $[\$v\$]$ are in the binarization[3]. This vertex set is a minimum vertex cover. Since all these operations are polynomial-time, this reduces the vertex cover problem to the minimum binarization problem, and therefore proves our main result, that the latter is NP-hard:

Theorem 1. *The minimum binarization problem is NP-hard.*

This theorem can be extended to show that the problem remains NP-hard even for grammars with a constant number of terminal symbols. To do so, note that we can reduce the minimum binarization problem for a grammar $G = (N, \Sigma, P, S)$ to the same problem for a grammar with a single terminal symbol x, $G_x = (N \cup \Sigma, \{x\}, P \cup P_\Sigma, S)$, where

$$P_\Sigma = \{a \to x \mid a \in \Sigma\}.$$

The grammar G_x treats each terminal symbol in G as a nonterminal, and then adds productions from each of those nonterminals to the single, fresh terminal symbol x. As each of these productions already has only one symbol in its right-hand side and cannot be split into smaller rules, they do not have any effect in binarizations. Thus, if we have a minimum binarization for G_x, removing these rules from it gives us a minimum binarization for G. This yields the following result:

Corollary 1. *The minimum binarization problem is NP-hard, even if the problem is restricted to CFGs where the number of terminal symbols is bounded by a constant $k \geq 1$.*

4. Inapproximability

As a side effect of our main result, an inapproximability bound for the minimum binarization problem can be obtained by using known inapproximability results for the minimum vertex cover problem on bounded-degree graphs. We will now prove the following theorem:

[3]This uses the fact that minimum grammars must be sensible binarizations. If instead we wish to use Lemma 1 directly, we can recover the vertex set by parsing every string of the form $\$v\$$ with any polynomial-time parsing algorithm such as CKY.

Theorem 2. *For every $\epsilon > 0$, it is NP-hard to approximate the minimum binarization problem within a factor of* $2575/2574 - \epsilon$.

Proof. We prove this result using reduction from the vertex cover problem for graphs $\mathcal{G} = (V, E)$ with maximum degree four and $|E| \geq |V|$. The problem of finding the minimum vertex cover of such a graph can be reduced to finding a minimum binarization of the grammar $G(\mathcal{G})$, as described in Section 3. We have also determined in that section that the size of such a minimum binarization is $2(6|V| + |E| + k)$, where k is the size of a minimum vertex cover of $\mathcal{G}$.

Since it has been shown by Berman and Karpinski [29] that it is NP-hard to approximate the minimum vertex cover for this set of graphs below the ratio 79/78, this means that it is also NP-hard to approximate the minimum binarization of their corresponding grammars $G(\mathcal{G})$ below a ratio of

$$\frac{6|V| + |E| + \frac{79}{78}k}{6|V| + |E| + k}.$$

Since we are restricting ourselves to graphs of degree not greater than 4, we know that $|E| \leq 2|V|$. This also means that each vertex can appear in at most four edges, so the size k of the minimum vertex cover must be at least $\frac{1}{4}|E| \geq \frac{1}{4}|V|$. Therefore, we can obtain a lower bound for the approximation ratio by setting $|E| = 2|V|$ and $k = \frac{1}{4}|V|$:

$$\frac{8|V| + \frac{79}{78}\frac{1}{4}|V|}{8|V| + \frac{1}{4}|V|} = \frac{2575}{2574}$$

This establishes that the minimum binarization problem is NP-hard to approximate within a factor of $2575/2574 - \epsilon$ for $\epsilon > 0$, and thus Theorem 2 is proved. □

By comparison, we note that the smallest grammar problem is known to be NP-hard to approximate within a factor of $8569/8568 - \epsilon$ [21].

5. Conclusion

We have shown that the problem of finding the minimum-size binarization of a CFG is NP-hard, and that every efficient approximation algorithm for this problem has an approximation ratio of at least 2575/2574 unless $P = NP$.

The presented hardness results not only hold for CFGs and the enriched formalisms that use them as a backbone (such as probabilistic CFGs or unification grammars), but also for a wide range of wider-coverage formalisms that are supersets of CFGs, like conjunctive grammars [9], binarizable synchronous context-free grammars [10], well-nested linear context-free rewriting systems [15], coupled context-free grammars [12] or regular tree grammars [13].

To see why this is so, we note that for conjunctive grammars and regular tree grammars the generalization is trivial, since the CFG $G(\mathcal{G})$ for a given graph $\mathcal{G}$ can be directly written as a conjunctive grammar or a regular tree grammar without using the extra features of these formalisms, and the proof still stands. For synchronous context-free grammars, we carry over the proof with a grammar that duplicates $G(\mathcal{G})$ in the source and target sides. In the case of coupled context-free grammars or well-nested linear context-free rewriting systems, we can again use $G(\mathcal{G})$ directly, but it is not so trivial that the result still holds because these formalisms allow for more flexibility in binarizations (since they allow reductions to be applied to non-contiguous symbols in a production rule). However, it can easily be checked that, in the particular case of the grammars $G(\mathcal{G})$ used in our proof, doing such reductions is always a suboptimal decision and will never lead to a minimum binarization, so the proof is still valid for these formalisms.

To our knowledge, these are the first published hardness results for the minimum binarization problem, and they are relevant for the computational linguistics and natural language processing community: grammar binarization is widely used both as a necessary requisite of some parsing algorithms and to improve the efficiency of others, and the size of the binarized grammar is known to heavily affect parsing accuracy.

Acknowledgements

This research has been partially funded by the Spanish Ministry of Economy and Competitiveness and FEDER (project TIN2010-18552-C03-02) and Xunta de Galicia (CN2012/008, CN2012/319, Rede Galega de Recursos Lingüísticos para unha Sociedade do Coñecemento).

I thank Miguel A. Alonso for his helpful comments on an early draft of this article.

References

[1] T. Kasami, An Efficient Recognition and Syntax Algorithm for Context-Free Languages, Technical Report AF-CRL-65-758, Air Force Cambridge Research Laboratory, 1965.

[2] D. H. Younger, Recognition and parsing of context-free languages in time n^3, Information and Control 10 (1967) 189–208.

[3] N. Chomsky, On certain formal properties of grammars, Information and Control 2 (1959) 137–167.

[4] M. Lange, H. Leiß, To CNF or not to CNF? An efficient yet presentable version of the CYK algorithm, Informatica Didactica 8 (2009).

[5] J. Earley, An efficient context-free parsing algorithm, Communications of the ACM 13 (1970) 94–102.

[6] D. J. Rosenkrantz, P. M. Lewis, Deterministic left corner parsing, in: Proceedings of the 11th Symposium on Switching and Automata Theory, SWAT'70, IEEE Computer Society, Washington, DC, USA, 1970, pp. 139–152.

[7] S. L. Graham, M. Harrison, W. L. Ruzzo, An improved context-free recognizer, ACM Trans. Program. Lang. Syst. 2 (1980) 415–462.

[8] G. Van Noord, An efficient implementation of the head-corner parser, Computational Linguistics 23 (1997) 425–456.

[9] A. Okhotin, Conjunctive grammars, Journal of Automata, Languages and Combinatorics 6 (2001) 519–535.

[10] L. Huang, H. Zhang, D. Gildea, K. Knight, Binarization of synchronous context-free grammars, Computational Linguistics 35 (2009) 559–595.

[11] C. Gómez-Rodríguez, M. Kuhlmann, G. Satta, D. J. Weir, Optimal reduction of rule length in linear context-free rewriting systems, in: Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Association for Computational Linguistics, Stroudsburg, PA, USA, 2009, pp. 539–547.

[12] G. Hotz, G. Pitsch, On parsing coupled-context-free languages, Theoretical Computer Science 161 (1996) 205–233.

[13] A. Koller, M. Kuhlmann, A generalized view on parsing and translation, in: Proceedings of the Twelfth International Conference on Parsing Technologies (IWPT), Association for Computational Linguistics, Stroudsburg, PA, USA, 2011, pp. 2–13.

[14] O. Rambow, G. Satta, Independent parallelism in finite copying parallel rewriting systems, Theoretical Computer Science 223 (1999) 87–120.

[15] C. Gómez-Rodríguez, M. Kuhlmann, G. Satta, Efficient parsing of well-nested linear context-free rewriting systems, in: Proceedings of Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL), Association for Computational Linguistics, Stroudsburg, PA, USA, 2010, pp. 276–284.

[16] X. Song, S. Ding, C.-Y. Lin, Better binarization for the CKY parsing, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP '08, Association for Computational Linguistics, Stroudsburg, PA, USA, 2008, pp. 167–176.

[17] C. Gómez-Rodríguez, M. A. Alonso, M. Vilares, On theoretical and practical complexity of TAG parsers, in: *Shuly Wintner (ed.),* Proceedings of FG 2006: The 11th conference on Formal Grammar, *volume of* FG Online Proceedings, CSLI publications, Stanford, CA, USA, 2006, pp. 87–101.

[18] H. Schmid, Efficient parsing of highly ambiguous context-free grammars with bit vectors, in: Proceedings of the 20th international conference on Computational Linguistics, COLING '04, Association for Computational Linguistics, Stroudsburg, PA, USA, 2004, pp. 162–168.

[19] J. DeNero, M. Bansal, A. Pauls, D. Klein, Efficient parsing for transducer grammars, in: Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, NAACL '09, Association for Computational Linguistics, Stroudsburg, PA, USA, 2009, pp. 227–235.

[20] D. Gildea, Grammar factorization by tree decomposition, Comput. Linguist. 37 (2011) 231–248.

[21] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, A. Shelat, The smallest grammar problem, IEEE Transactions on Information Theory 51 (2005) 2554–2576.

[22] W. Rytter, Application of Lempel–Ziv factorization to the approximation of grammar-based compression, Theor. Comput. Sci. 302 (2003) 211–222.

[23] J. A. Storer, T. G. Szymanski, Data compression via textual substitution, J. ACM 29 (1982) 928–951.

[24] A. V. Aho, M. S. Lam, R. Sethi, J. D. Ullman, Compilers: Principles, Techniques, and Tools (2nd Edition), Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[25] A. Nijholt, Context-Free Grammars: Covers, Normal Forms, and Parsing, volume 93 of *Lecture Notes in Computer Science*, Springer-Verlag, 1980.

[26] R. Leermakers, How to cover a grammar, in: Proceedings of the 27th annual meeting on Association for Computational Linguistics, ACL '89, Association for Computational Linguistics, Stroudsburg, PA, USA, 1989, pp. 135–142.

[27] J. E. Hopcroft, J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 1979.

[28] R. M. Karp, Reducibility among combinatorial problems, in: R. Miller, J. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, 1972, pp. 85–103.

[29] P. Berman, M. Karpinski, On some tighter inapproximability results, further improvements, Tech. Rep. TR98-065, Electronic Colloquium on Computational Complexity (ECCC) 5 (1998).