On the Binary-Alphabet Complexity of the Assembly Index: NP-Completeness of ASI-DEC and Consequences for SLP and SGP Variants

Piotr Masierak *

Posted Date: 27 February 2026

doi: 10.20944/preprints202602.1964.v1

Keywords: assembly theory; assembly index; grammar-based compression; computational complexity; information theory; complexity measures

 doi:10.20944/preprints202602.1964.v1

Article

On the Binary-Alphabet Complexity of the Assembly Index: NP-Completeness of ASI-DEC and Consequences for SLP and SGP Variants

Piotr Masierak

Łukaszyk Patent Attorneys, ul. Głowackiego 8, 40-052 Katowice, Poland; pmasierak@patent.pl

Abstract

We study the canonical string-based Assembly Index (ASI), defined as the minimum number of binary concatenations needed to construct a target word under full reuse. NP-completeness of ASI-DEC over general finite alphabets and an equivalence between ASI plans and straight-line programs (SLPs) under the same size convention has been established. We emphasize that all transfers between decision variants are effected by explicit polynomial-time mappings and (where needed) an explicit reparameterization of the threshold by an absolute constant or a simple affine function. The remaining technical obstacle for the binary alphabet is that a naive encoding reduction may allow an optimizer to exploit "cross-boundary" substrings created by overlaps of codewords. We give a fully self-contained binary-alphabet proof: We construct an explicit self-synchronizing (comma-free) codebook of 17 fixed-length binary codewords and prove a boundary-normalization lemma showing that optimal plans can be assumed aligned to codeword boundaries. This yields a polynomial reduction from fixed-alphabet ASI-DEC to binary ASI-DEC, proving NP-completeness over $\{0,1\}$. Using the recalled ASI–SLP equivalence (with a short proof for completeness), we obtain NP-completeness of binary SLP-DEC. We additionally provide an explicit, fully formal translation between our binary-rule counting convention and the standard SGP size measure (sum of right-hand side lengths), showing that the NP-completeness classification transfers to common one-string SGP/SLP decision variants over $\{0,1\}$.

Keywords: assembly theory; assembly index; grammar-based compression; computational complexity; information theory; complexity measures

1. Introduction, Motivation, and Related Work

Assembly Theory proposes the assembly index as a measure of compositional complexity, defined as the minimum number of binary composition (concatenation) steps needed to build an object from primitives. In [1], we formalized the canonical decision problem ASI-DEC and proved NP-completeness for general (unbounded) alphabets by an explicit correspondence between assembly plans and straight-line programs (SLPs).

In applications, the underlying alphabet is often fixed and small (e.g., bits, DNA, or small symbol sets), and reductions that rely on growing alphabets are not directly applicable. For the classical Smallest Grammar Problem (SGP) / minimum-size SLP problem, early NP-hardness proofs use an alphabet whose size grows with the instance (e.g., [3,4]). Whether such hardness persists for fixed alphabets was long open; Casel et al. proved NP-completeness for every fixed alphabet of size at least 17 [2]. For the binary alphabet, several works explicitly note that adapting known constructions is nontrivial and that the status was unclear in that line of research [2,5].

In the grammar-based compression literature, the Smallest Grammar Problem (SGP) is typically stated for a grammar generating a single string, but the size can be defined in several (polynomially related) ways: counting nonterminals, counting productions, or counting the total number of symbols

 doi:10.20944/preprints202602.1964.v1

on all right-hand sides. An SLP is a particularly restricted grammar where every production has the binary-concatenation form $X \to YZ$ (or a terminal), and it is the standard normal form used in algorithmics on grammar-compressed strings [6]. For single-string grammars, optimizing over general grammars versus SLPs is interchangeable up to polynomial transformations by binarization of right-hand sides. This paper works with the most direct assembly-theoretic size measure: The number of binary concatenations (i.e., the number of rules of the form $X \to YZ$). We therefore state the transfer to SLP-DEC/SGP-DEC under this convention, and we also include an explicit lemma (Section 7) showing how to convert to common SGP/SLP size conventions by an affine or polynomial-time reparameterization of the threshold. Hence the NP-completeness classification is unaffected by the choice among standard size measures.

We give a self-contained NP-completeness proof for ASI-DEC over the binary alphabet $\{0,1\}$, using a synchronizing block code and a constant-overhead dictionary gadget that prevents cross-boundary reuse. We then formalize a bridge theorem showing that, under the same binary-concatenation size measure, ASI-DEC is parsimoniously equivalent to the binary SLP-DEC decision problem (and thus to the one-string SGP-DEC variants under standard size conventions). Consequently, the paper provides an NP-completeness proof for the binary SLP decision problem in the corresponding size convention, and clarifies the relation to standard grammar-size measures.

Section 2 recalls the model and decision problems. Sections 3–5 develop the coding and normalization lemmas. Section 6 proves binary NP-hardness and NP-completeness for ASI-DEC. Section 7.1 derives NP-completeness for binary SLP (and notes the implication for one-string SGP variants) and discusses implications.

2. Model, Problems, and Background

We work in the unconstrained binary-join model: At any time one may concatenate any two previously constructed strings. Reuse is free and unlimited. Terminals are available at zero cost.

Definition 1. *For a word $w \in \Sigma^+$, $\mathrm{ASI}(w)$ is the minimum number of binary concatenations needed to construct w from the terminals Σ.*

Definition 2. ASI-DEC*: Given (w,k), decide whether $\mathrm{ASI}(w) \le k$.*

Definition 3. *A Straight-Line Program (one-string grammar) SLP for a single target word w is a context-free grammar that generates exactly w and whose nonterminal productions are binary concatenations $X \to YZ$. The size $|G|$ is the number of such binary productions.*[1] *Let $\mathrm{SLP}(w)$ be the minimum size of an SLP generating w.*

Lemma 1. *For every word w, $\mathrm{ASI}(w) = \mathrm{SLP}(w)$.*

Proof. We use the same size convention as in the main paper [1]: The size of an SLP is the number of binary concatenation productions $X \to YZ$ (terminal productions $X \to a$ are not counted).

SLP $\Rightarrow$ plan. Given an SLP with m binary productions, evaluate its nonterminals in a topological order. Each production $X \to YZ$ corresponds to one assembly step that concatenates the already-built strings of Y and Z to obtain the string of X. Hence $\mathrm{ASI}(w) \le m$.

Plan $\Rightarrow$ SLP. Given an assembly plan with m concatenation steps, create one nonterminal for each intermediate object and one production $X \to YZ$ for each concatenation. Choose the nonterminal of the final object as the start symbol. This yields an SLP of size m, hence $\mathrm{SLP}(w) \le m$.

Therefore $\mathrm{ASI}(w) = \mathrm{SLP}(w)$. □

Lemma 2. *Over any alphabet,* ASI-DEC *belongs to NP.*

[1] If a venue counts size differently (e.g., adds a constant per terminal declaration), all results remain valid after a polynomial-time threshold translation; see Section 7.2.

 doi:10.20944/preprints202602.1964.v1

Proof. A certificate is an explicit concatenation DAG (or the list of concatenation steps with pointers to previously built objects) of size at most k. A verifier reconstructs all intermediate strings and checks that the final one equals w. Each intermediate string has length at most $|w|$, hence verification runs in polynomial time. □

Remark 1. *It is known that the Smallest Grammar Problem remains NP-complete for a fixed alphabet of size at least 17 ; see, e.g., Casel et al. for fixed alphabets. This implies NP-completeness of* ASI-DEC *over some fixed alphabet* $\Sigma_0 = \{a_1, \ldots, a_{17}\}$ *via Lemma 1.*

Implications.

The corollary places binary SLP (and one-string SGP variants) on the same worst-case complexity footing as the classical unbounded-alphabet setting. Unless P = NP, there is no polynomial-time exact algorithm for producing a minimum-size SLP for a given binary word. In addition, approximation-related hardness statements for grammar-based compression are sensitive to the chosen objective and reduction notion, and thus should be interpreted with care for fixed alphabets. In our setting, the bridge identifies the objective value on binary inputs under the "number of binary concatenation rules" convention; consequently, any statement formulated solely in terms of this objective carries over verbatim between ASI and SLP on $\{0,1\}$.

The purpose of this paper is to remove the remaining binary-alphabet technicality by giving a stand-alone encoding-and-normalization argument that prevents "bit-level shortcuts".

3. Why Binary Requires Extra Work: Cross-Boundary Reuse

A naive replacement of each high-level symbol by a short binary string can change the optimal assembly cost, because a binary optimizer is free to reuse any repeated bit-pattern—even one that crosses the intended symbol boundaries. This is the core obstacle for reductions to $\{0,1\}$.

Suppose (for illustration only) that two distinct symbols are encoded naively as $A = 010$ and $B = 101$. Then the binary string for AB is

$$AB \mapsto 010101 = (01)(01)(01),$$

so an optimizer may first build 01 once, then reuse it to build 0101, and finally build 010101. This kind of reuse does not correspond to any legal reuse of whole symbols at the high level: It arises purely from misalignment (a reuse that "slides" across the $A|B$ boundary). The same phenomenon appears even more starkly if one considers longer concatenations where a cross-boundary fragment happens to repeat in several places, creating spurious low-level building blocks.

To preserve NP-hardness under alphabet reduction, we need an encoding h such that no cross-boundary substring can become a reusable building block unless it corresponds to an actual concatenation of whole symbols. In our setting this is achieved by two coupled ingredients: (i) a self-synchronizing (comma-free) codebook C whose codeword boundaries are detectable in the raw bitstream, and (ii) a boundary normalization lemma showing that every optimal (or near-optimal) plan/SLP can be transformed, without increasing cost, into one whose intermediate objects are concatenations of whole codewords.

Intuitively, our markers force every true boundary to contain the anchor pattern 11110000 (end-marker followed by start-marker). Any misaligned substring that crosses such a boundary is "tied" to that unique boundary and therefore cannot reappear elsewhere, so it cannot be useful for reuse; the normalization lemma then formalizes the elimination of all such one-off fragments.

 doi:10.20944/preprints202602.1964.v1

4. A Concrete Self-Synchronizing Codebook

Fix $\Sigma_0 = \{a_1, \ldots, a_{17}\}$. We give an explicit injective encoding

$$c : \Sigma_0 \to \{0,1\}^{16}, \qquad c(a_i) = 0000\, b_i\, 1111,$$

where $b_i \in \{0,1\}^8$ is a payload chosen to satisfy a simple constraint.

Definition 4. *A fixed-length codebook $C \subseteq \{0,1\}^L$ satisfies **(SYNC)** if no codeword can be formed by a nontrivial suffix-prefix overlap of two codewords: For all $u, v, w \in C$ and all $1 \le t \le L-1$,*

$$\mathrm{suf}_{L-t}(u)\ \mathrm{pre}_t(v) \neq w.$$

In particular, since all codewords have the same fixed length L, condition extbf(SYNC) rules out any ambiguous parsing caused by shifted overlaps; hence every word in C^* has a unique decomposition into length-L codewords.

Proposition 1. *Fix $L = 16$. Let $C = \{c(a_1), \ldots, c(a_{17})\} \subseteq \{0,1\}^L$ be the explicit codebook from Appendix A, where each*

$$c(a_i) = 0000\, b_i\, 1111$$

and the payloads $b_i \in \{0,1\}^8$ are chosen so that: (i) all b_i are distinct, (ii) no b_i contains 0000 or 1111 as a substring, (iii) each b_i starts with 1 and ends with 0. Then:

1. *C satisfies* (SYNC) *(Definition 4);*
2. *the boundary marker $B := 11110000$ occurs in any $x \in C^*$ only across genuine codeword boundaries, and at every genuine boundary it occurs exactly once.*

Proof. Item (1) follows from Lemma A10, which proves (SYNC) for the explicit codebook.

For item (2), note that by construction no payload b_i contains 1111 or 0000 as an internal substring, and b_i starts with 1 and ends with 0. Hence within any single codeword $0000\, b_i\, 1111$ the substring 11110000 cannot occur (it would require 1111 followed immediately by 0000 inside the codeword). In a concatenation of codewords, the only place where 1111 is immediately followed by 0000 is at the junction between two consecutive codewords, i.e., at a genuine codeword boundary. Therefore $B = 11110000$ occurs exactly once per boundary and nowhere else. □

Remark 2. *Let $u = 0000\, b\, 1111$ and $v = 0000\, b'\, 1111$ be two codewords in C (so $|u| = |v| = L = 16$). Consider any crossing length-L substring s obtained by taking a nonempty suffix of u and a nonempty prefix of v (so s starts strictly inside u and ends strictly inside v). If s were itself a codeword, then it would have to start with the prefix marker 0000 and end with the suffix marker 1111. But because s crosses the boundary between u and v, any occurrence of the marker 0000 at the beginning of s forces the boundary pattern*

$$B := 1111\,0000 = 11110000$$

to appear internally in s (namely at the junction between the ending 1111 of u and the beginning 0000 of v). By Proposition 1, no codeword contains B internally. Hence no such crossing substring can be a codeword, which is exactly the kind of overlap excluded by (SYNC).

5. Boundary Normalization

Extend c homomorphically to words: For $w = a_{i_1} \cdots a_{i_m} \in \Sigma_0^+$ let

$$h(w) = c(a_{i_1}) \cdots c(a_{i_m}) \in C^*.$$

Definition 5. *A string $x \in \{0,1\}^*$ is aligned if $x \in C^*$. An intermediate object in an assembly plan (or nonterminal in an SLP) is aligned if its value is aligned; otherwise it is misaligned.*

 doi:10.20944/preprints202602.1964.v1

Lemma 3. *Let $x \in C^*$. Any assembly plan that produces x (equivalently, any SLP for x) can be transformed in polynomial time into an equivalent assembly plan in which no intermediate object starts in one codeword block of x and ends in the next. Equivalently, every intermediate object that spans multiple C-blocks is aligned (i.e., lies in C^*), without increasing the number of concatenation steps.*

Proof. We argue in the assembly-plan model and use the plan–SLP equivalence (Lemma 1) only as a convenient notation. Let P be an assembly plan for $x \in C^*$, and let G be the corresponding SLP obtained by naming each intermediate object by a nonterminal and each concatenation step by a rule $X \to YZ$. Thus every intermediate object in P corresponds to some nonterminal X with value $\mathrm{val}(X)$.

Call a nonterminal (equivalently, intermediate object) misaligned if its value is not in C^*. Fix the boundary marker $B = 11110000$ from Proposition 1. For intuition, see Appendix B for a small worked example.

We distinguish two cases for a misaligned value $\mathrm{val}(X) \notin C^*$.

(i) Boundary-crossing case. Suppose $\mathrm{val}(X)$ starts in one C-block of x and ends in the next (i.e., it crosses at least one genuine codeword boundary in the unique C-factorization of x). Then $\mathrm{val}(X)$ contains the boundary marker $B = 11110000$ as a substring (Proposition 1), and moreover this occurrence of B is tied to a unique boundary position of x. Consequently, $\mathrm{val}(X)$ cannot occur as a substring of x at two different positions.

In a straight-line program generating a *single* output string, if a nonterminal X is referenced in at least two distinct places in the derivation tree (i.e., appears at least twice on right-hand sides), then its value $\mathrm{val}(X)$ must occur in the output as a substring at least twice (at the corresponding two intervals). Therefore, in the boundary-crossing case, X can be referenced at most once.

(ii) Non-boundary-crossing case. If $\mathrm{val}(X)$ does not cross any codeword boundary of x, then it is entirely contained within a single C-block of x. Such nonterminals are harmless for our reduction because they cannot "bridge" two adjacent codeword blocks, and in particular cannot bridge the distinguished boundary between the fixed prefix D and the suffix $h(w)$ later in Lemma 6.

Thus, every misaligned nonterminal that *crosses a boundary* is used at most once, and can be eliminated without increasing the number of concatenation steps by the inlining procedure below.

Whenever such a boundary-crossing misaligned nonterminal X occurs as one child in a binary rule, i.e., in a production $P \to XY$ or $P \to YX$, replace that single occurrence by the (binary) derivation tree rooted at X ("inline" the unique use of X), and delete X from the grammar. Because X is used at most once, this does not increase the number of concatenation rules and does not change the generated string. Repeating this process eliminates all boundary-crossing misaligned nonterminals. In particular, the resulting plan is aligned in the sense that every intermediate object that spans multiple C-blocks lies in C^*, and no intermediate object starts in one block and ends in the next. The total number of concatenation steps does not increase. □

6. Reduction to the Binary Alphabet

Let D be a fixed binary dictionary prefix containing all codewords twice:

$$D := c(a_1)c(a_2)\cdots c(a_{17})\, c(a_1)c(a_2)\cdots c(a_{17}) \in C^*.$$

Let $m := |D|_C$ be the number of C-blocks in D (here $m = 34$), and define the prefix arrangement constant

$$T_{\mathrm{arr}} := m - 1 = 33.$$

This is the minimum number of binary concatenations needed to arrange m already-available codeword blocks into D; it is a constant independent of the input instance.

Given an instance (w, k) of ASI-DEC over Σ_0, define

$$f(w,k) := (w', k') \quad \text{where} \quad w' := D\,h(w) \in \{0,1\}^*, \quad k' := k + T_{\mathrm{arr}}.$$

 doi:10.20944/preprints202602.1964.v1

In the ASI model the only terminals are the bits $\{0,1\}$, so producing a codeword $c(a_i)$ of length $L = 16$ from terminals requires a constant number of concatenations.

For example, one can build 0^{16} in 15 concatenations by iterative doubling, and then obtain any 16-bit word by reusing suitable power-of-two blocks; in particular, each fixed codeword $c(a_i)$ can be built from terminals using at most $L - 1 = 15$ concatenations (e.g., by concatenating its 16 terminal bits in a binary tree). Therefore there is an explicit absolute bound $T_{\mathrm{cw}} \leq 17 \cdot 15$ for constructing all 17 codewords (and in fact a much smaller bound is possible with reuse, since the same intermediate blocks can be shared across codewords).

Let T_{cw} be the minimum number of concatenations required to build all 17 distinct codewords from $\{0,1\}$ (with full reuse), and let $T_{\mathrm{const}} := T_{\mathrm{cw}} + T_{\mathrm{arr}}$. Both T_{cw} and T_{const} are absolute constants independent of the instance (w,k). Our use of T_{arr} isolates the unavoidable arrangement cost of the fixed prefix D once the block words are available; any additional cost for constructing the blocks themselves is absorbed into the constant offset T_{cw} and does not affect polynomial-time reducibility or NP-completeness.

The mapping is computable in time $O(|w|)$.

Lemma 4. *Let $D \in C^*$ be a concatenation of m codeword blocks. Any assembly plan (or SLP) that produces D given the m codeword blocks as available reusable subobjects, using only binary concatenation has at least $m - 1$ concatenation steps. In particular, for our fixed D with $m = 34$ blocks we have $T_{\mathrm{arr}} = 33$.*

Proof. Initially one has at least m separate components (the m blocks). Each binary concatenation can reduce the number of components by at most 1. To obtain a single component equal to D therefore requires at least $m - 1$ concatenations. □

Lemma 5. *If $\mathrm{ASI}(w) \leq k$ over Σ_0, then $\mathrm{ASI}(w') \leq k'$ over $\{0,1\}$.*

Proof. Take an assembly plan for w of cost at most k. First build D using an optimal plan of cost T. Now simulate the plan for w on the encoded symbols by replacing each terminal a_i with the already-built block $c(a_i)$ and performing the same sequence of concatenations. This yields $w' = D\,h(w)$ using at most $T_{\mathrm{arr}} + k = k'$ concatenations (plus a constant T_{cw} to build the fixed codewords from terminals). □

Lemma 6. *If $\mathrm{ASI}(w') \leq k'$ over $\{0,1\}$, then $\mathrm{ASI}(w) \leq k$ over Σ_0.*

Proof. Let P be an optimal assembly plan for w' of cost at most k'. By Lemma 1 view P as an SLP for w', and apply Lemma 3 to obtain a normalized plan $\widehat{P}$ of cost at most k' in which no intermediate object crosses a codeword boundary between C-blocks (and, in particular, no intermediate crosses the boundary between the prefix D and the suffix $h(w)$).

In $\widehat{P}$, alignment implies that no intermediate object can cross the boundary between the fixed prefix D and the suffix $h(w)$: Otherwise it would contain the boundary marker $B = 11110000$ internally, contradicting Proposition 1 and the normalization invariant. Equivalently, in the unique C-factorization of the output $D\,h(w)$, every node of the concatenation DAG yields a contiguous block interval that lies either entirely within the first $m = |D|_C = 34$ blocks (the prefix region) or entirely within the remaining blocks (i.e., the blocks forming $h(w)$) (the suffix region).

Even if all m prefix blocks were available as atoms, arranging them into the exact prefix string D requires at least $T_{\mathrm{arr}} = m - 1$ binary concatenations (Lemma 4). Hence $\widehat{P}$ contains at least T_{arr} concatenation steps whose outputs lie in the prefix region.

 doi:10.20944/preprints202602.1964.v1

Delete from $\widehat{P}$ a set of T_{arr} prefix-region concatenations that witnesses this unavoidable arrangement cost. The remaining plan has cost at most

$$k' - T_{\text{arr}} = k$$

and still produces the suffix $h(w)$ (possibly reusing any already-constructed codeword blocks as atoms). This is valid because after normalization no intermediate object used to construct a suffix-region node depends on concatenations whose outputs lie solely in the prefix region; any such dependency would require an intermediate spanning the prefix–suffix boundary, which is excluded by the normalization invariant.

Finally, project back to Σ_0 by replacing each block $c(a_i)$ by a_i. Alignment ensures every concatenation remains valid and produces w. Hence $\text{ASI}(w) \leq k$. □

Theorem 1. ASI-DEC *over the binary alphabet* $\{0,1\}$ *is NP-complete.*

Proof. Membership in NP is Lemma 2. NP-hardness follows from Lemmas 5 and 6, which show

$$\text{ASI}(w) \leq k \iff \text{ASI}(D\,h(w)) \leq k + T_{\text{arr}},$$

a polynomial-time many-one reduction from fixed-alphabet NP-complete ASI-DEC (Remark 1) to binary ASI-DEC. □

7. Binary SLP Is NP-Complete via ASI

7.1. SLP vs. SGP and Size Conventions

Lemma 7. *Let G be a context-free grammar that generates exactly one string w and whose productions have right-hand sides of length at least* 1. *There is a polynomial-time transformation producing an equivalent SLP G' that generates exactly w such that the number of binary concatenation productions in G' is $O(|G|)$ under any standard size measure $|G|$ that counts the total number of symbols on right-hand sides (equivalently: Total RHS length).*

Proof. Replace every production $X \to Y_1 Y_2 \cdots Y_r$ with $r \geq 3$ by introducing fresh nonterminals and a binary parse chain: $X \to Y_1 Z_2$, $Z_2 \to Y_2 Z_3$, …, $Z_{r-1} \to Y_{r-1} Y_r$. This preserves the generated string and increases the number of binary productions by at most $r - 1$. Summed over all productions, the blow-up is linear in the total RHS length. □

Remark 3. *If a venue defines grammar size differently (e.g., counting nonterminals, productions, or total RHS length), Lemma 7 implies a polynomial relationship between thresholds. Thus, NP-completeness under our binary-concatenation size implies NP-completeness for the standard SGP/SLP decision variants by polynomial-time reductions.*

We recall (proved in [1]) the equivalence between ASI plans and SLPs under the same size convention; for completeness we include a short proof.

Theorem 2 (Bridge theorem)**.** *For every binary word $x \in \{0,1\}^+$,* $\text{ASI}(x) = \text{SLP}(x)$.

Proof. We use the same size convention as in [1]: SLP size is the number of binary concatenation rules $X \to YZ$.

SLP $\Rightarrow$ ASI. Evaluate the SLP in a topological order; each rule $X \to YZ$ is one assembly concatenation producing the string of X. Thus $\text{ASI}(x) \leq \text{SLP}(x)$.

ASI $\Rightarrow$ SLP. From an assembly plan with m concatenations, create one nonterminal per intermediate object and one production per concatenation; the final object is the start symbol. This yields an SLP of size m, so $\text{SLP}(x) \leq \text{ASI}(x)$. □

 doi:10.20944/preprints202602.1964.v1

Corollary 1. *Under the size convention of this paper (SLP size = number of binary concatenation rules), the decision problem* SLP-DEC *over* $\{0,1\}$ *is NP-complete. Moreover, by standard polynomial-time reductions between common one-string grammar size measures, the corresponding one-string* SGP-DEC *variants over* $\{0,1\}$ *are NP-complete as well.*

Proof. NP-membership is standard (certificate: An SLP of size at most k). NP-hardness follows by the identity reduction $(x,k) \mapsto (x,k)$ from binary ASI-DEC using Theorem 2:

$$\mathrm{ASI}(x) \le k \iff \mathrm{SLP}(x) \le k. \quad \square$$

7.2. From Binary-Rule Size to Standard SGP Size (Sum of RHS Lengths)

In the grammar-based compression literature, the Smallest Grammar Problem (SGP) for a single target string is often measured by the total length of all right-hand sides (RHS) of productions, sometimes with minor variations (e.g., whether to include the start symbol, terminal productions, or separators). To avoid any ambiguity when comparing to the state of the art, we make the following explicit.

Definition 6. *Let G be a context-free grammar that generates exactly one string. Define*

$$|G|_{\mathrm{RHS}} := \sum_{(X \to \alpha) \in P} |\alpha|,$$

i.e., the sum of lengths of all right-hand sides (counting terminals and nonterminals as symbols). The corresponding decision problem is: Given (w,K), is there such a grammar G with $L(G) = \{w\}$ and $|G|_{\mathrm{RHS}} \le K$.

Lemma 8. *Let G be an SLP in binary form (productions $X \to YZ$ and $X \to a$). If m is the number of binary productions $X \to YZ$ and r is the number of terminal productions $X \to a$, then*

$$|G|_{\mathrm{RHS}} = 2m + r.$$

In particular, over the binary alphabet $\{0,1\}$, we may assume $r \le 2$, hence $|G|_{\mathrm{RHS}} = 2m + O(1)$.

Proof. Each binary production contributes 2 to the RHS length; each terminal production contributes 1. Over $\{0,1\}$, terminal productions can be shared globally (at most one for 0 and one for 1) without affecting the derived string, so we may assume $r \le 2$. $\square$

Lemma 9. *Let G be a grammar generating exactly one string, with $|G|_{\mathrm{RHS}} = K$. There exists, in polynomial time, an equivalent SLP G' generating the same string such that*

$$\#\textit{binary rules in } G' \le K \quad \textit{and} \quad |G'|_{\mathrm{RHS}} \le O(K).$$

Proof. Because G generates a single string, we can (i) delete unreachable and unproductive nonterminals, and (ii) break any RHS of length $\ell \ge 2$ into a chain of $\ell - 1$ binary concatenation rules by introducing fresh nonterminals, preserving the generated string. Each created binary rule corresponds to consuming at least one symbol from some original RHS; thus the total number of created binary rules is at most $\sum |\alpha| = K$. Terminal occurrences can be factored through at most two terminal rules over $\{0,1\}$. The construction is standard and runs in polynomial time. $\square$

Corollary 2. *Over the binary alphabet $\{0,1\}$, NP-completeness of* SLP-DEC *under binary-rule counting implies NP-completeness of the one-string SGP decision problem under the standard size measure $|G|_{\mathrm{RHS}}$ from Definition 6.*

 doi:10.20944/preprints202602.1964.v1

Proof. By Lemma 8, an SLP of m binary rules yields a grammar of RHS-size at most $2m+2$. Hence $(w,m) \in$ SLP-DEC $\Rightarrow (w, 2m+2) \in \mathrm{SGP}_{\mathrm{RHS}}$-DEC. Conversely, by Lemma 9, any RHS-size-K one-string grammar yields an SLP with at most K binary rules, so $(w,K) \in \mathrm{SGP}_{\mathrm{RHS}}$-DEC $\Rightarrow (w,K) \in$ SLP-DEC. Thus the problems are polynomial-time interreducible with explicit threshold mappings, and NP-completeness is preserved. □

8. Additional Remarks

Our main result concerns the binary alphabet $\{0,1\}$, where rich encodings are possible. At the opposite extreme, for a unary alphabet $\{0\}$ there is exactly one word of each length, namely 0^N. In this case, the structural content of an optimal assembly plan is purely arithmetical: Every binary concatenation corresponds to adding lengths. Consequently, the minimum number of concatenations needed to build 0^N from 0 coincides with the length $\ell(N)$ of the shortest addition chain for N (see, e.g., Theorem 3.1 in [7] and OEIS A003313 for values).

It is important to distinguish two input models. In the standard ASI-DEC formulation studied in this paper, the input is the word itself. For the unary alphabet this means the input size is $|0^N| = N$ (an explicit representation). In contrast, the classical "addition-chain decision" problem typically takes N in binary, so the input size is $\Theta(\log N)$ (a succinct representation). Thus, even if computing $\ell(N)$ is nontrivial from a number-theoretic perspective, algorithms exponential in $\log N$ may still be polynomial in N and therefore do not imply NP-hardness for the explicit-string unary case.

For completeness we note that one can define a succinct unary variant of ASI-DEC: Given (N,k) with N in binary, decide whether $\mathrm{ASI}(0^N) \le k$. Under the plan–addition-chain correspondence, this succinct unary problem is essentially equivalent to the classical shortest addition-chain decision problem. We do not study this succinct variant here; our NP-completeness result is for the explicit-string model and becomes nontrivial precisely when $|\Sigma| \ge 2$.

9. Conclusions

We established the computational complexity of the Assembly Index decision problem in the binary setting. Our main result (Theorem 1) shows that ASI-DEC over the alphabet $\{0,1\}$ is NP-complete.

The NP-hardness proof is based on an explicit fixed-length codebook $C \subseteq \{0,1\}^{16}$ satisfying (SYNC) (Definition 4 and Proposition 1) and the induced homomorphic encoding $h(\cdot)$ into C^*. The key technical step is the boundary-normalization lemma (Lemma 3), which allows us to assume that any assembly plan (equivalently, any SLP under our size convention) producing a word in C^* contains no boundary-crossing misaligned intermediate objects, i.e., no intermediate object starts in one C-block and ends in the next. This rules out cross-boundary reuse that could otherwise invalidate the reduction.

Using the fixed binary dictionary prefix

$$D := c(a_1)\cdots c(a_{17})\, c(a_1)\cdots c(a_{17}) \in C^*$$

and the associated constant prefix-arrangement cost $T_{\mathrm{arr}} = |D|_C - 1$, we obtain a polynomial-time many-one reduction

$$(w,k) \longmapsto (D\,h(w),\, k + T_{\mathrm{arr}}),$$

formalized in Lemmas 5 and 6. In particular, the threshold mapping is explicit and involves only the additive constant T_{arr}.

Finally, by the bridge theorem (Theorem 2) equating assembly plans and straight-line programs over binary inputs under the same binary-concatenation size convention, the NP-completeness result transfers to the corresponding binary decision variants for SLP/SGP via the standard threshold comparison and the explicit size mappings developed in Section 7.

This paper focuses on decision problems and NP-completeness statements. Although the bridge theorem suggests related consequences for exact optimization variants under the same convention, a

 doi:10.20944/preprints202602.1964.v1

fully explicit optimization-level treatment would require a separate discussion of objective conventions and reduction notions, and is left for future work.

A promising direction for future research concerns approximation questions in grammar-based compression over fixed alphabets. Our Boundary-Normalization Lemma shows that, for strings in C^* produced by our self-synchronizing encoding, boundary-crossing misaligned intermediates can be eliminated without increasing the objective under the size convention used here. This suggests that the self-synchronizing structure of the underlying codebook may serve as a useful restriction when analyzing (or designing) approximation algorithms for SLP/SGP variants over the binary alphabet via such encodings. In particular, it would be interesting to understand to what extent approximation guarantees (and known lower bounds) for grammar-based compressors are preserved under these structured binary encodings. We do not pursue approximation results in this paper.

Funding: This research received no external funding.

Acknowledgments: I thank my partners Szymon Łukaszyk and Wawrzyniec Bieniawski for inspiring the topic of this publication and their clarifications, formal corrections and improvements.

Conflicts of Interest: The author Piotr Masierak was employed by the company Łukaszyk Patent Attorneys. The author declares that the research was conducted in the absence of any commercial or financial relationship that could be construed as a potential conflict of interest.

Appendix A. Codebook Table

Binary-alphabet reductions are sometimes criticized as "non-constructive" if they only assert the existence of a synchronizing code. Here we provide an explicit 17-word codebook (Table A1) so that every step of the reduction (including checking (SYNC)) can be verified mechanically.

We use fixed-length codewords of length $L = 16$ with a unique prefix marker $P = 0000$ and a unique suffix marker $S = 1111$:

$$c(a_i) \;=\; P\, b_i\, S \;=\; 0000\, b_i\, 1111, \qquad b_i \in \{0,1\}^8.$$

The chosen middle blocks b_i satisfy three simple constraints: (i) all b_i are distinct, (ii) b_i contains no substring 0000 and no substring 1111, (iii) b_i starts with 1 and ends with 0 (so that the concatenations Pb_i and b_iS do not create internal occurrences of 0000 or 1111 across the marker boundaries). These conditions ensure that P occurs only as the prefix of a codeword and S occurs only as the suffix of a codeword. The absence of internal occurrences of 0000 and 1111 in Table A1 can be verified mechanically (e.g., by a short script).

Table A1. Explicit synchronizing codebook $c(a_i) = 0000\, b_i\, 1111$ of length 16.

Symbol	b_i (length 8)	$c(a_i)$ (length 16)
a_1	10001000	0000100010001111
a_2	10001010	0000100010101111
a_3	10001100	0000100011001111
a_4	10001110	0000100011101111
a_5	10010010	0000100100101111
a_6	10010100	0000100101001111
a_7	10010110	0000100101101111
a_8	10011000	0000100110001111
a_9	10011010	0000100110101111
a_{10}	10011100	0000100111001111
a_{11}	10100010	0000101000101111
a_{12}	10100100	0000101001001111
a_{13}	10100110	0000101001101111
a_{14}	10101000	0000101010001111
a_{15}	10101010	0000101010101111
a_{16}	10101100	0000101011001111
a_{17}	10101110	0000101011101111

 doi:10.20944/preprints202602.1964.v1

Lemma A10. *The codebook in Table A1 satisfies (SYNC).*

Proof. Let C be the codebook. Consider any nontrivial suffix–prefix crossing yz of two codewords $u, v \in C$ that has length $L = 16$. If yz were a codeword, it would have to begin with the marker $P = 0000$ and end with the marker $S = 1111$.

Because yz starts strictly inside u (nontrivial suffix) or ends strictly inside v (nontrivial prefix), at least one of the following holds:

- the first position of yz lies strictly inside u, hence the prefix 0000 would have to occur internally in u, or
- the last position of yz lies strictly inside v, hence the suffix 1111 would have to occur internally in v.

However, by the construction constraints above, 0000 appears in a codeword only as its first four bits, and 1111 appears only as its last four bits. Therefore neither case is possible, so $yz \notin C$. This is exactly (SYNC). □

Appendix B. A Small Worked Example Illustrating Normalization

Let $w = a_1a_2a_1$. Then $h(w) = c(a_1)c(a_2)c(a_1)$ is a concatenation of three 16-bit blocks. Any substring that crosses a block boundary contains the unique boundary pattern 11110000. Hence a misaligned intermediate string (one that starts inside one block and ends inside the next) cannot occur in two different places of $h(w)$, so it cannot be beneficially reused. Lemma 3 formalizes this intuition by eliminating all such one-off misaligned nonterminals by inlining.

References

1. Piotr Masierak. Computational Complexity of Determining the Assembly Index. *IPI Letters*, 4(1):9–12, 2026. doi:10.59973/ipil.315. Available at: https://ipipublishing.org/index.php/ipil/article/view/315.
2. Katrin Casel, Henning Fernau, Serge Gaspers, Benjamin Gras, and Markus L. Schmid. On the Complexity of the Smallest Grammar Problem over Fixed Alphabets. *Theory of Computing Systems*, 65:344–409, 2021. doi:10.1007/s00224-020-10013-w.
3. Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, April Rasala, Amit Sahai, and Abhi Shelat. Approximating the Smallest Grammar: Kolmogorov Complexity in Natural Models. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC 2002)*, pages 792–801. ACM, 2002. doi:10.1145/509907.510021.
4. James A. Storer and Thomas G. Szymanski. Data Compression via Textual Substitution. *Journal of the ACM*, 29(4):928–951, 1982. doi:10.1145/322344.322346. (Extended abstract: *The Macro Model for Data Compression*, STOC 1978.)
5. Danny Hucke, Markus Lohrey, and Carl Philipp Reh. The Smallest Grammar Problem Revisited. In *String Processing and Information Retrieval (SPIRE 2016)*, pages 35–49. Springer, 2016. doi:10.1007/978-3-319-46049-9_4.
6. Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012. doi:10.1515/gcc-2012-0016.
7. Wawrzyniec Bieniawski, Andrzej Tomski, Szymon Łukaszyk, Piotr Masierak, and Szymon Tworz. Assembly Theory – Formalizing Assembly Spaces, Discovering Patterns and Bounds. *Preprints.org*, 2025. Preprint 202409.1581, Version 12. doi:10.20944/preprints202409.1581.v12. Posted: 29 December 2025. Available at: https://www.preprints.org/manuscript/202409.1581.