IPI Letters

ISSN 2976 - 730X
IPI Letters 2026,Vol 4 (1):9-12
https://doi.org/10.59973/ipil.315

Received: 2026-01-02
Accepted: 2026-01-15
Published: 2026-01-20

Article

Computational Complexity of Determining the Assembly Index

Piotr Masierak [1,*]

[1] Łukaszyk Patent Attorneys, ul. Głowackiego 8, 40-052 Katowice, Poland

* Corresponding author: pmasierak@patent.pl

Abstract - The assembly index of assembly theory quantifies the minimal number of composition steps required to construct an object from elementary components. The study proves that the decision version of the assembly index problem is NP-complete, through an explicit correspondence between assembly plans and straight-line grammars. This correspondence implies that the optimization version of the assembly index problem inherits NP- and APX-hardness from the classical smallest grammar problem. The study provides complete, self-contained proofs for both decision and optimization variants of the assembly index problem. These results establish that computing or approximating the assembly index is computationally intractable, placing it within the same complexity class as grammar-based compression.

Keywords Assembly theory; Assembly index; Grammar-based-compression; Computational complexity; Information theory; Complexity measures; NP-completeness.

1 Introduction

Assembly theory (AT) [1-3] studies how complex structures emerge from simpler components via reuse of intermediates. The assembly index (ASI) quantifies this generative complexity as the minimal number of joining steps required to construct a structure from elementary and intermediate components. Intuitively, the ASI captures how deeply an object is embedded in a hierarchy of possible assembly pathways. Although this concept has been widely discussed in theoretical and experimental contexts, its precise computational characterization has remained open. It was conjectured [4] that the problem of determining the assembly index is[1] NP-complete. An attempt to prove this conjecture was offered in [5]. However, this proof was shown to violate the principles of AT by a predefined assembly space involving only a set of predefined assembly steps [6]. In this work, we consider the general, string-based formulation of the ASI with binary concatenation and assembly spaces over strings with free terminals and unlimited reuse.

2 Methods

The smallest grammar problem (SGP) was shown to be NP-complete [7], with additional results on APX-hardness and bounds on approximation ratios. Furthermore, connections

[1] Formally, a problem *is not* a complexity class, but belongs to a complexity class.

between SLPs and Lempel–Ziv factorization were analyzed [8] and the hardness results were extended to fixed alphabets and Chomsky normal form grammars [9].
We shall show that determining the ASI is computationally equivalent to finding the shortest straight-line program (SLP) generating w, a concept well known in grammar-based compression. This equivalence allows us to transfer established hardness results from the smallest grammar problem (SGP) to the ASI, connecting these previously separate research areas.
We shall prove that the decision version of the ASI problem (ASI-DEC) is NP-complete and that its optimization version (ASI-OPT) is NP- and APX-hard. These findings place the ASI within the same theoretical framework as grammar-based compression, showing that computing or approximating ASI(w) is computationally intractable unless $P = NP$.

3 Results

Let Σ be a finite alphabet and $w \in \Sigma^+$ a word. The initial pool contains all letters of Σ, the assembly step involves the concatenation ($\circ$) of exactly two existing words (letters or words previously formed), and each previously formed word may be reused arbitrarily many times. The goal is to construct the word w, and the cost of achieving the goal is the number of assembly steps. The ASI(w) is the minimal number of assembly steps required to construct the word w.

Definition 1 (ASI-DEC)**.** *Given a pair* (w, k) *with* $w \in \Sigma^+$ *and* $k \in \mathbb{N}$*, decide whether* $\mathrm{ASI}(w) \leq k$*.*

The problem is existential: the input does not include the assembly plan but only asks whether such a plan exists. The NP witness corresponds to such a plan, and the verifier checks it in polynomial time by reconstructing the concatenation sequence. The distinction between the assembly steps problem and the assembly index problem is that the former concerns verifying a specific assembly sequence within a bounded number of steps, while the latter, formalized in Definition 1, is existential, asking only whether such a sequence exists. Accordingly, its NP witness is the assembly plan itself.

A straight-line program (SLP) is a context-free grammar generating exactly one word w, with rules of the form $X \to YZ$ or $X \to a$. Its size is the number of concatenation rules $X \to YZ$.

Definition 2 (SLP-DEC)**.** *Given a pair* (w, K)*, decide whether there exists an SLP of size less than or equal to K generating w. This problem is NP-complete [7-9].*

Lemma 1. *For any word w, the minimal number of assembly steps equals the minimal number of concatenation rules in an SLP generating w, that is* $\mathrm{ASI}(w) = \mathrm{SLP}(w)$*.*

Proof. Each assembly plan of k concatenations corresponds to an SLP of size k, where every step $U \circ V$ becomes a production $X \to UV$. Conversely, every SLP of size K yields a K-step assembly plan by expanding its rules in topological order. Both count the same number of concatenations. □

For example, let $w = 01010$ over binary alphabet $\Sigma = \{0, 1\}$ (the initial pool). An optimal assembly plan containing three steps S_i may have the form

$$\begin{aligned} S_1 &:= 0 \circ 1 = 01; \\ S_2 &:= S_1 \circ 0 = 01 \circ 0 = 010; \\ S_3 &:= S_2 \circ S_1 = 010 \circ 01 = 01010 = w. \end{aligned}$$

corresponding to the SLP of size $K = 3$

$$\begin{aligned} R_1 &\to 0 \circ 1 \quad \text{(Rule 1 corresponds to step } S_1\text{)}; \\ R_2 &\to R_1 0 \quad \text{(Rule 2 corresponds to step } S_2\text{)}; \\ R_3 &\to R_2 R_1 \quad \text{(Rule 3 corresponds to step } S_3\text{)}. \end{aligned}$$

Lemma 2. ASI-DEC $\in$ *NP.*

Proof. Let the input be (w, k), where $w \in \Sigma^+$ has length $n := |w|$ and $k \in \mathbb{N}$ is encoded in binary. We shall show that there exists a polynomial-time verifier whose running time is polynomial in the input size $n + \log_2 k$.

Any construction of a word of length n from single letters using binary concatenations needs at most $n - 1$ concatenation steps: starting from pieces of total length 1, each concatenation increases the length of the newly created word by at least 1, and the final length is n. Hence, if $k \geq n - 1$ then the instance is trivially YES. Otherwise, $k \leq n - 2$, so any feasible witness plan uses at most $t \leq k = O(n)$ steps.

A witness is an assembly plan given as a sequence of t steps $X_i := U_i \circ V_i$ for $i = 1, 2, \ldots, t$, where $t \leq k$. Each U_i, V_i is specified *by pointers/indices* either to: (i) a terminal symbol $a \in \Sigma$ (constant-size encoding), or (ii) one of the previously constructed words X_j with $j < i$. Thus, the witness size is $O(t \log_2 t) = O(n \log_2 n)$ bits.

The verifier reconstructs all intermediate strings X_i in increasing order of i and checks that the last constructed string equals w. Each intermediate string has length at most n, hence the total verification time is $O(t \cdot n) \subseteq O(n^2)$ because $t \leq k \leq n - 1$ in the nontrivial cases, which is polynomial in n and hence also polynomial in the input size $n + \log_2 k$. □

Theorem 1. ASI-DEC $\in$ NP-complete *for any finite alphabet* Σ.

Proof. From Lemma 2 we know that ASI-DEC $\in$ NP. For NP-hardness, we reduce SLP-DEC to ASI-DEC: by Lemma 1, for every instance (w, K), we define $(w, k := K)$, which leads to

$$(w, K) \in \text{SLP-DEC} \iff \text{SLP}(w) \leq K \iff \text{ASI}(w) \leq K \iff (w, k) \in \text{ASI-DEC}.$$

The reduction is identity-based and computable in $O(|w|)$ time, proving the NP-hardness. □

The reduction is purely existential and does not require an explicit plan or grammar. It shows that the existence of a grammar of size lower than or equal K is equivalent to the existence of an assembly plan of cost lower than or equal K. Therefore, determining ASI(w) lies in the same computational class as finding the smallest SLP.

Theorem 1 shows that determining whether a word can be assembled within k concatenations is, therefore, in general, computationally intractable unless $P = NP$.

For a given word w, we define the ASI optimization version:

Definition 3 (ASI-OPT)**.** *Given $w \in \Sigma^+$, minimize the number of its assembly steps to reach* ASI(w).

Let SLP(w) denote the minimal number of concatenation rules in an SLP generating w.

Theorem 2. *The optimization version of the ASI problem given by Definition 3 is NP-hard and APX-hard. Computing* ASI(w) *cannot be done in polynomial time, and no polynomial-time approximation scheme exists unless $P = NP$.*

Proof. Let SLP-OPT denote the optimization problem of finding the smallest SLP generating w. SLP-OPT is NP-hard and APX-hard [7-9]. By Lemma 1, ASI(w) = SLP(w) for all w, so the identity mapping $f(w) = w$ defines a cost-preserving polynomial reduction. Therefore, both NP- and APX-hardness results transfer directly to ASI-OPT. □

4 Conclusions

We have shown that determining the ASI is NP-complete (decision version) and NP-/APX-hard (optimization version) for an arbitrary finite alphabet. The equality ASI(w) = SLP(w) places ASI within the same complexity framework as the SGP. Consequently, computing or approximating ASI(w) is as difficult as finding the smallest grammar generating w.

Acknowledgments

I thank my partners Wawrzyniec Bieniawski and Szymon Łukaszyk for 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.

References

[1] Marshall SM, Murray ARG, Cronin L. A probabilistic framework for identifying biosignatures using Pathway Complexity. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences. 2017 Dec;375(2109):20160342. Available from: https://royalsocietypublishing.org/doi/10.1098/rsta.2016.0342.

[2] Marshall SM, Moore DG, Murray ARG, Walker SI, Cronin L. Formalising the Pathways to Life Using Assembly Spaces. Entropy. 2022 Jun;24(7):884. Available from: https://www.mdpi.com/1099-4300/24/7/884.

[3] Sharma A, Cz´egel D, Lachmann M, Kempes CP, Walker SI, Cronin L. Assembly theory explains and quantifies selection and evolution. Nature. 2023 Oct;622(7982):321-8. Available from: https://www.nature.com/articles/s41586-023-06600-9.

[4] Łukaszyk S, Bieniawski W. Assembly Theory of Binary Messages. Mathematics. 2024 May;12(10):1600. Available from: https://www.mdpi.com/2227-7390/12/10/1600

[5] Kempes CP, Lachmann M, Iannaccone A, Matthew Fricke G, Redwan Chowdhury M, Walker SI, et al. Assembly theory and its relationship with computational complexity. npj Complexity. 2025 Sep;2(1):27. Available from: https://www.nature.com/articles/s44260-025-00049-9

[6] Łukaszyk S. On the "Assembly Theory and its Relationship with Computational Complexity". IPI Letters. 2025 Jan:1-6. Available from: https://ipipublishing.org/index.php/ipil/article/view/157

[7] Charikar M, Lehman E, Liu D, Panigrahy R, Prabhakaran M, Sahai A, et al. The smallest grammar problem. IEEE Transactions on Information Theory. 2005;51(7):2554-76. Available from: https://web.cs.ucla.edu/œsahai/work/web/2005%20Publications/TransOnInfoTheory2005.pdf

[8] Rytter W. Application of Lempel–Ziv factorization to the approximation of grammarbased compression. Theoretical Computer Science. 2003 Jun;302(1-3):211-22. Available from: https://linkinghub.elsevier.com/retrieve/pii/S0304397502007776

[9] Casel K, Fernau H, Gaspers S, Gras B, Schmid ML. On the Complexity of the Smallest Grammar Problem over Fixed Alphabets. Theory of Computing Systems. 2021 Feb; 65(2):344-409. Available from: http://link.springer.com/10.1007/s00224-020-10013-w