The Journal of Open Source Software

assembly-theory: Open, Reproducible Calculation of Assembly Indices

Devansh Vimal [1], **Garrett Parzych** [1,2], **Olivia M. Smith** [1,3], **Devendra Parkar** [1,2], **Holly Bergen** [1,2], **Joshua J. Daymude** [1,2], **and Cole Mathis** [1,3¶]

1 Biodesign Center for Biocomputing, Security and Society, Arizona State University, United States **2** School of Computing and Augmented Intelligence, Arizona State University, United States **3** School of Complex Adaptive Systems, Arizona State University, United States ¶ Corresponding author

DOI: 10.21105/joss.09318

Software

- Review
- Repository
- Archive

Editor: Gabriela Alessio Robles

Reviewers:

- @abhishektiwari
- @amcandio

Submitted: 07 August 2025
Published: 23 December 2025

Summary

We present `assembly-theory`, an open-source, high-performance library for computing *assembly indices* of covalently bonded molecular structures. This is a key complexity measure of *assembly theory*, a recent theoretical framework quantifying selection across diverse systems, most importantly chemistry. `assembly-theory` is designed for researchers and practitioners alike, providing (i) extensible, high-performance Rust implementations of assembly index calculation algorithms, (ii) comprehensive tests and benchmarks against which current and future algorithmic improvements can be evaluated, and (iii) Python bindings to support integration with existing computational pipelines.

Background

Assembly theory (AT) is a recently developed body of theoretical and empirical work characterizing selection in diverse physical systems (Sharma et al., 2023; Walker et al., 2024). In AT, objects are entities that are finite, distinguishable, decomposable, and persistent in time. AT characterizes objects by their *assembly index*, the minimum number of recursive subconstructions required to construct the object starting from a given set of building blocks (Jirasek et al., 2024; Seet et al., 2025). To date, AT has most commonly been applied to molecular chemistry, where bonds are the basic building blocks and the quantity of interest is the *molecular assembly index* (MA); see Figure 1 for an example. MA can be measured for covalently-bonded molecules using standard analytical techniques such as tandem mass spectrometry as well as infrared and nuclear magnetic resonance spectroscopy (Jirasek et al., 2024), enabling a novel approach to life detection (Marshall et al., 2021). It has also been proposed in methods to generate novel therapeutic drugs, identify environmental pollutants, and gain new insights into evolutionary history (Kahana et al., 2024; Liu et al., 2021).

Statement of Need

Computing MA efficiently remains a challenge. In general, exact MA calculation is NP-hard (Kempes et al., 2024). Previous software to compute MA have been approximate, closed-source, platform-dependent, or written in languages rarely used by the broader scientific community. The original software to compute a split-branch approximation of MA (an upper bound on the exact value) was written in C++ and depended on the MSVC compiler, making it difficult to deploy to non-Windows machines (Marshall et al., 2021). Machine learning methods only provide approximate MA values (Gebhard et al., 2022). The more

recent `assembly_go` computes MA exactly but is written in Go and implements a somewhat naive algorithm, yielding prohibitively slow performance even on mid-size molecules (Jirasek et al., 2024). Finally, the latest `assemblycpp-v5` C++ implementation achieves significant performance milestones through an improved branch-and-bound approach, but lacks parallelism, has significant readability and maintenance barriers, and until recently was not publicly available for comparison or verification by the community (Seet et al., 2025).

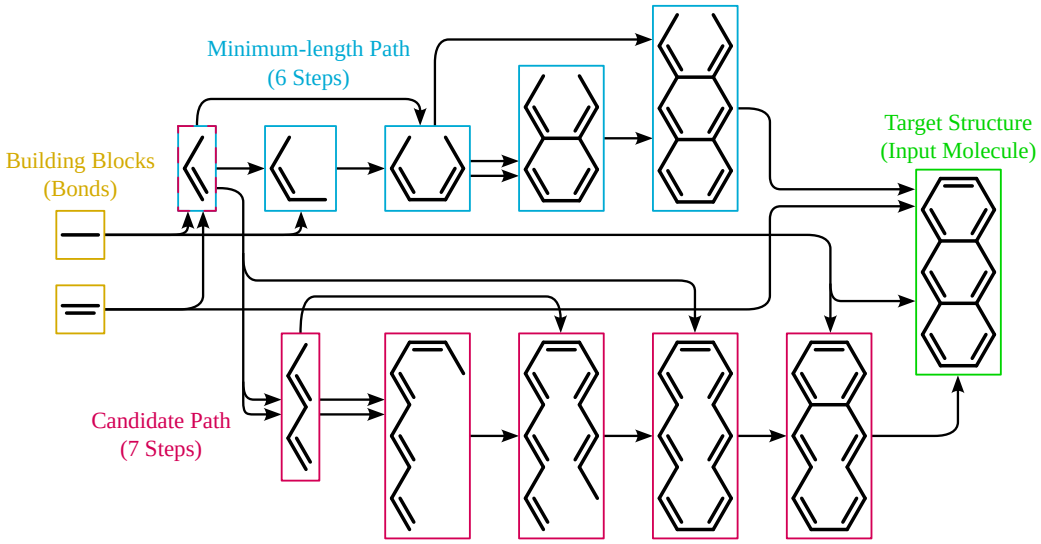

Figure 1: *Assembly Pathways for Anthracene*. Starting with bonds as building blocks (yellow), a joining operation yields progressively larger structures by combining any two compatible structures that have already been constructed (arrows). These intermediate structures must obey valence rules but otherwise do not have to be physically accessible or chemically synthesizable. There may be many assembly pathways from building blocks to a target structure—in this case, Anthracene (green)—but the length of any shortest such pathway (blue) is that structure's assembly index.

With `assembly-theory`, we provide an open-source, fully documented, extensible, and high-performance library for assembly index calculation. It moves beyond an implementation of a single algorithm, instead acting as a framework and source of ground truth within which current and future algorithmic approaches can be validated and compared. The main implementation is written in Rust, which we chose for its cross-platform support, memory-safety, performant runtime, convenient parallelism, and integrated testing and documentation (Perkel, 2020). We also leverage modern Rust tooling to provide native Python bindings, enabling ease of use for scientific practitioners and integration with existing Python cheminformatics libraries.

Functionality and Usage Examples

`assembly-theory` is available as a standalone executable, a Rust crate, and a Python package. Here, we provide usage examples of each; in the next section, we describe testing and benchmarking functionality.

Standalone Executable

Rust provides the `cargo` build system and dependency manager for compilation, testing, benchmarking, documentation, and packaging. From source, build the executable with:

```
> cargo build --release
```

Simply pass this executable a `.mol` file to compute that molecule's assembly index:

```
> ./target/release/assembly-theory data/checks/anthracene.mol  # 6
```

A full list of options for customizing the assembly index calculation procedure is obtained with:

```
> ./target/release/assembly-theory --help
```

Rust Crate

The publicly released `assembly-theory` Rust crate is hosted on crates.io and can be included in a broader Rust project with:

```
> cargo add assembly-theory
```

Complete documentation of the crate is available on docs.rs; a simple usage example is:

```rust
use assembly_theory::{
    assembly::index,
    loader::parse_molfile_str
};

// Load a molecule from a `.mol` file.
let path = PathBuf::from(format!("./data/checks/anthracene.mol"));
let molfile = fs::read_to_string(path)?;
let anthracene = parse_molfile_str(&molfile).expect("Parsing failure.");

// Compute the molecule's assembly index.
assert_eq!(index(&anthracene), 6);
```

Python Package

We use the `pyo3` crate and `maturin` to create a Python package that mirrors the Rust crate's public functionality. This Python package can be installed from PyPI in the usual way:

```
> pip install assembly-theory
```

This package is designed for compatibility with `RDKit`, the standard Python library for cheminformatics (*RDKit*, 2024). Molecules can be loaded and manipulated using the `rdkit.Chem.Mol` class and then passed to our functions for assembly index calculation:

```python
import assembly_theory as at
from rdkit import Chem

# Get a mol block from a molecule's SMILES representation.
anthracene = Chem.MolFromSmiles("c1ccc2cc3ccccc3cc2c1")
anthracene = Chem.MolToMolBlock(anthracene)

# Calculate the molecule's assembly index.
at.index(anthracene)  # 6
```

Tests and Benchmarks

`assembly-theory` includes test and benchmark suites for software validation and performance evaluation, respectively. Both use curated reference datasets representing different classes of molecules, chosen for their structural diversity and approachable runtime on commodity hardware. These reference data are sampled from:

- GDB-13, a database of enumerated chemical structures containing Carbon, Hydrogen, Nitrogen, Oxygen, Sulfur, and Chlorine that are constrained only by valence rules and quantum mechanics (Blum & Reymond, 2009).

- GDB-17, an extension of GDB-13 which includes additional nuclei such as the halogens Flourine and Iodine (Ruddigkeit et al., 2012).
- KEGG COMPOUND, a database of small molecules, biopolymers, and other biologically relevant substances (Kanehisa, 2019; Kanehisa et al., 2023; Kanehisa & Goto, 2000).
- COCONUT, a database of natural products (secondary metabolites) offering a rich source of evolved chemical complexity (Chandrasekhar et al., 2025; Sorokina et al., 2021).

The `assembly-theory` test suite (run with `cargo test`) contains unit tests validating internal functionality and integration tests verifying the calculation of correct assembly indices for all molecules in our reference datasets. Each reference dataset contains an `ma-index.csv` file with ground truth assembly indices calculated using `assemblycpp-v5` (Seet et al., 2025).

Our benchmark suite (run with `cargo bench`) evaluates the performance of each granular phase of assembly index calculation over entire reference datasets. We leverage the `criterion` Rust crate to automatically collect detailed timing statistics and create performance reports.

Performance Evaluation

Table 1 shows a comparison of MA calculation times across the three existing open-source implementations on four curated reference datasets. Details of this benchmark and its reference datasets can be found in `paper/README`. `assembly_go` is orders of magnitude slower than the other two implementations, and the performance gap widens as molecule sizes increase (`checks` and `coconut_55`). Between `assemblycpp-v5` and `assembly-theory`, our `assembly-theory` implementation performs 1.76–2.27x faster on average for all but the smallest molecules (`gdb13_1201`), where `assemblycpp-v5` achieves a mean speedup of 1.52x.

Table 1: Mean benchmark execution times for `assembly_go` (6ec034f), `assemblycpp-v5` (f920903), and `assembly-theory` (v0.6.0) across reference datasets. The benchmark times the MA calculation of all molecules in a given dataset in sequence, excluding the time required to parse and load `.mol` files into internal molecular graph representations. Each benchmark was run on a Linux machine with a 5.7 GHz Ryzen 9 7950X CPU (16 cores) and 64 GB of memory. `assembly_go` and `assembly-theory` are parallel implementations and used all 16 cores, while `assemblycpp-v5` is serial and used only one. Means and 95% confidence intervals are reported over 20 samples per software–dataset pair, except those marked with an $*$ which have prohibitively long runtimes and thus ran only once.

	assembly_go	assemblycpp-v5	assembly-theory
gdb13_1201	0.938 s ± 6.00%	**0.107 s** ± 0.19%	0.163 s ± 0.53%
gdb17_200	46.523 s ± 1.00%	0.318 s ± 0.24%	**0.181 s** ± 0.71%
checks	215.194 s ± 0.55%	0.053 s ± 0.88%	**0.025 s** ± 0.24%
coconut_55	1.34 h*	0.345 s ± 0.26%	**0.152 s** ± 0.45%

Algorithmically, the default MA search strategies of `assemblycpp-v5` and our `assembly-theory` are currently very similar. Our speedup on larger molecules is likely due primarily to parallelism, which `assemblycpp-v5` lacks. However, we emphasize that `assembly-theory` offers advantages beyond performance, including native Python interoperability, detailed documentation, and a modular architecture that enables the implementation and comparison of current and future algorithmic approaches within the same framework without language-based confounding factors.

Availability and Governance

`assembly-theory` is available as a source code repository on GitHub, as a Rust crate on crates.io, and as a Python package on PyPI. Following the standard practice for Rust projects, `assembly-theory` is dual-licensed under the MIT and Apache-2.0 licenses. Contributing

guidelines and project governance are described in our `README`. Benchmarks and performance evaluations supporting this manuscript are available on Zenodo (AgentElement et al., 2025).

Author Contributions

DV was the primary software developer (architecture, command line interface, molecule representations, unit tests, parallelism, performance engineering). GP, DV, and CM formalized the core algorithm design. GP and HB implemented the algorithm's bounding strategies. DP and DV implemented the `.mol` file parser. CM and JJD implemented the Python interface. OMS curated all reference datasets and assembly index ground truths with input from CM. JJD created the integration tests and benchmarks. JJD conducted and analyzed the benchmark shown in Table 1. JJD and CM wrote the paper.

Acknowledgements

GP and JJD are supported in part by NSF award CCF-2312537. DV, OMS, and CM acknowledge support from the ASU Biodesign Institute.

References

AgentElement, Daymude, J., Mathis, C., Garrett-Pz, Smith, O., Parkar, D., & Bergen, H. (2025). *DaymudeLab/assembly-theory* (joss-final). Zenodo. https://doi.org/10.5281/zenodo.16764412

Blum, L. C., & Reymond, J.-L. (2009). 970 Million Druglike Small Molecules for Virtual Screening in the Chemical Universe Database GDB-13. *Journal of the American Chemical Society*, *131*(25), 8732–8733. https://doi.org/10.1021/ja902302h

Chandrasekhar, V., Rajan, K., Kanakam, S. R. S., Sharma, N., Weißenborn, V., Schaub, J., & Steinbeck, C. (2025). COCONUT 2.0: A comprehensive overhaul and curation of the collection of open natural products database. *Nucleic Acids Research*, *53*(D1), D634–D643. https://doi.org/10.1093/nar/gkae1063

Gebhard, T. D., Bell, A. C., Gong, J., Hastings, J. J. A., Fricke, G. M., Cabrol, N., Sandford, S., Phillips, M., Warren-Rhodes, K., & Baydin, A. G. (2022). Inferring molecular complexity from mass spectrometry data using machine learning. *Machine Learning and the Physical Sciences Workshop at NeurIPS 2022*, 1–7.

Jirasek, M., Sharma, A., Bame, J. R., Mehr, S. H. M., Bell, N., Marshall, S. M., Mathis, C., MacLeod, A., Cooper, G. J. T., Swart, M., Mollfulleda, R., & Cronin, L. (2024). Investigating and Quantifying Molecular Complexity Using Assembly Theory and Spectroscopy. *ACS Central Science*, *10*(5), 1054–1064. https://doi.org/10.1021/acscentsci.4c00120

Kahana, A., MacLeod, A., Mehr, H., Sharma, A., Carrick, E., Jirasek, M., Walker, S. I., & Cronin, L. (2024). *Constructing the Molecular Tree of Life using Assembly Theory and Mass Spectrometry* (No. 2408.09305). arXiv. https://doi.org/10.48550/arxiv.2408.09305

Kanehisa, M. (2019). Toward understanding the origin and evolution of cellular organisms. *Protein Science*, *28*(11), 1947–1951. https://doi.org/10.1002/pro.3715

Kanehisa, M., Furumichi, M., Sato, Y., Kawashima, M., & Ishiguro-Watanabe, M. (2023). KEGG for taxonomy-based analysis of pathways and genomes. *Nucleic Acids Research*, *51*(D1), D587–D592. https://doi.org/10.1093/nar/gkac963

Kanehisa, M., & Goto, S. (2000). KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Research*, *28*(1), 27–30. https://doi.org/10.1093/nar/28.1.27

Kempes, C., Walker, S. I., Lachmann, M., & Cronin, L. (2024). *Assembly Theory and its Relationship with Computational Complexity* (No. 2406.12176). arXiv. https://doi.org/10.48550/arXiv.2406.12176

Liu, Y., Mathis, C., Bajczyk, M. D., Marshall, S. M., Wilbraham, L., & Cronin, L. (2021). Exploring and mapping chemical space with molecular assembly trees. *Science Advances*, *7*(39), eabj2465. https://doi.org/10.1126/sciadv.abj2465

Marshall, S. M., Mathis, C., Carrick, E., Keenan, G., Cooper, G. J. T., Graham, H., Craven, M., Gromski, P. S., Moore, D. G., Walker, Sara. I., & Cronin, L. (2021). Identifying molecules as biosignatures with assembly theory and mass spectrometry. *Nature Communications*, *12*(1), 3033. https://doi.org/10.1038/s41467-021-23258-x

Perkel, J. M. (2020). Why scientists are turning to Rust. *Nature*, *588*(7836), 185–186. https://doi.org/10.1038/d41586-020-03382-2

RDKit: Open-source cheminformatics. (2024). https://doi.org/10.5281/zenodo.591637

Ruddigkeit, L., Van Deursen, R., Blum, L. C., & Reymond, J.-L. (2012). Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17. *Journal of Chemical Information and Modeling*, *52*(11), 2864–2875. https://doi.org/10.1021/ci300415d

Seet, I., Patarroyo, K. Y., Siebert, G., Walker, S. I., & Cronin, L. (2025). Rapid Exploration of the Assembly Chemical Space of Molecular Graphs. *Journal of Chemical Information and Modeling*. https://doi.org/10.1021/acs.jcim.5c01964

Sharma, A., Czégel, D., Lachmann, M., Kempes, C. P., Walker, S. I., & Cronin, L. (2023). Assembly theory explains and quantifies selection and evolution. *Nature*, *622*(7982), 321–328. https://doi.org/10.1038/s41586-023-06600-9

Sorokina, M., Merseburger, P., Rajan, K., Yirik, M. A., & Steinbeck, C. (2021). COCONUT online: Collection of Open Natural Products database. *Journal of Cheminformatics*, *13*(1), 2. https://doi.org/10.1186/s13321-020-00478-9

Walker, S. I., Mathis, C., Marshall, S., & Cronin, L. (2024). Experimentally measured assembly indices are required to determine the threshold for life. *Journal of The Royal Society Interface*, *21*(220), 20240367. https://doi.org/10.1098/rsif.2024.0367