IBIMA Publishing
Communications of International Proceedings
https://ibimapublishing.com/p-articles/46SP/2025/4642125/
Vol. 2025 (37), Article ID 4642125

Performance Comparison of graph-tool and NetworkX on Web Graph Domain Subgraphs: A Common Crawl Analysis*

Krystian MAGDZIARZ and Stanisław SKRZYPECKI

Military University of Technology, Warsaw, Poland

Correspondence should be addressed to: Krystian MAGDZIARZ, krystian.magdziarz@student.wat.edu.pl

* Presented at the 46th IBIMA International Conference, 26-27 November 2025, Ronda, Spain

Abstract

Efficient graph processing is critical for web-scale analysis, yet practitioners lack empirical guidance for library selection on real-world data. We present a comprehensive performance comparison of graph-tool and NetworkX on Common Crawl web graph data, focusing on domain-level subgraph analysis. Through systematic benchmarking of seven core operations across thousands of domain subgraphs, we challenge the assumption that C++ libraries with Python bindings always outperform pure Python implementations. Our results reveal operation-dependent performance patterns: NetworkX excels in graph traversal operations (2.5-4.6× faster for connected components, shortest path, degree distribution) and community detection (7.5× faster), while graph-tool dominates computationally intensive algorithms (35× faster betweenness centrality, 4× faster clustering coefficient). Memory usage differs significantly, with NetworkX maintaining consistent 900-970 MB baseline versus graph-tool's operation-dependent overhead reaching 2.5 GB. The domain-based decomposition methodology enables statistical analysis across diverse website structures, revealing that optimal library choice depends critically on specific operations, graph size, and available system resources rather than blanket performance assumptions.

Keywords: graph processing, performance benchmarking, web graphs, Common Crawl, domain subgraphs, domain decomposition, NetworkX, graph-tool

Introduction

Graph analysis of web-scale networks is essential for understanding online information flow, link structure, and network topology (Meusel, 2015; Kil et al., 2009). This is particularly important from a cybersecurity perspective, especially in developing new methods for phishing detection (Tan et al., 2020; Li et al., 2024; Bilot, Geis, and Hammi, 2022; Castillo et al., 2007; E. Frąszczak and D. Frąszczak, 2024), where understanding web graph structure and link patterns can reveal malicious websites. The evolving landscape of cyber threats, including sophisticated attacks and disinformation campaigns, requires continuous innovation in digital resilience and technological capabilities (Chmielewski, 2024). When conducting such analyses, researchers and practitioners face a critical decision when selecting graph processing libraries. This decision requires empirical evidence from real-world web data rather than synthetic benchmarks.

Motivation

While existing benchmarks compare graph libraries on synthetic or small-scale datasets, real-world web graph analysis presents unique challenges:

Cite this Article as: Krystian MAGDZIARZ and Stanisław SKRZYPECKI, Vol. 2025 (37) "Performance Comparison of graph-tool and NetworkX on Web Graph Domain Subgraphs: A Common Crawl Analysis " Communications of International Proceedings, Vol. 2025 (37), Article ID 4642125, https://doi.org/10.5171/2025.4642125

- **Scale**: Web graphs contain millions of pages and billions of links
- **Heterogeneity**: Different domains exhibit vastly different link structures
- **Domain-level analysis**: Understanding individual website characteristics within larger graphs
- **Efficiency**: Need for representative metrics without processing entire graphs

Our work addresses this gap by introducing domain-based subgraph decomposition and systematically evaluating library performance on real Common Crawl web data.

Graph Processing Libraries

NetworkX (*NetworkX — NetworkX Documentation* 2025) is the de facto standard for graph analysis in Python, offering intuitive APIs and extensive documentation. Its pure-Python implementation prioritizes usability over performance.

graph-tool (*Graph-Tool* 2025) leverages C++ and the Boost Graph Library for high-performance graph processing. It uses efficient property maps and compiled algorithms but has a steeper learning curve.

Existing Benchmarks

While several benchmark studies have compared Python graph processing libraries (*Benchmark of Popular Graph/Network Packages V2* 2020; *Benchmark of Popular Graph/Network Packages* 2019), significant gaps remain in the literature. Existing benchmarks have predominantly evaluated performance on large monolithic graphs rather than domain-level subgraphs and have rarely utilized real-world web graph data for validation. Notably, graph-tool, currently recognized as one of the fastest Python graph processing libraries (Af'Ally and Fitriyani, 2024), has been underrepresented in comparative studies. Additionally, comprehensive statistical analysis with significance testing and effect sizes has been largely absent from published results. Our work addresses these limitations by evaluating graph-tool alongside NetworkX within the Python ecosystem, utilizing authentic web graph data from Common Crawl rather than synthetic datasets, introducing domain-based subgraph decomposition methodology for website-level analysis, providing rigorous statistical analysis with significance testing and effect sizes, and examining structural heterogeneity across different web domains.

Methodology

This study employs a systematic benchmarking approach to compare two graph processing libraries: NetworkX (*NetworkX — NetworkX Documentation* 2025) and graph-tool (*Graph-Tool* 2025) with graph_tool Python binding. We utilize Common Crawl WAT (Web Archive Transformation) (*Common Crawl - Open Repository of Web Crawl Data* 2025) files, which provide large-scale web graph data with authentic link structures and domain characteristics. We partition the web graph into domain-level subgraphs, where each domain represents an individual website's internal link structure. For each benchmark, we measure execution time, peak memory usage, CPU utilization, and I/O operations for both libraries across all domain subgraphs. Results are aggregated across all domains with significance testing (t-tests, effect sizes) to ensure robust conclusions. We verify that both libraries use equivalent algorithms for fair comparison, documenting any unavoidable differences.

Benchmark Framework

We developed a modular benchmark framework implementing the adapter pattern for unified library interface, resource monitoring (time, memory, CPU, I/O), statistical analysis (t-tests, effect sizes) and visualization. This methodology allows us to evaluate library performance in realistic scenarios while maintaining experimental rigor and reproducibility.

Dataset

We use Common Crawl WAT files to extract web link graphs. For this study, we utilize the March 2024 Common Crawl dataset (CC-MAIN-2024-10), from which we extracted 1 million edges. This dataset represents a medium-scale web graph suitable for comprehensive benchmarking. In our previous work, we demonstrated how to construct a high-performance scraping cluster for building large-scale web datasets (Magdziarz and D. Frąszczak, 2022), which can be used to create current datasets reflecting up-to-date web structures.

Common Crawl provides web archive data in WARC (Web ARChive) format. We utilize WAT (Web Archive Transformation) files, which contain extracted metadata from crawled pages.

Each WAT file contains JSON records with the following hierarchical structure:

```
{
```

```
"Envelope": {
  "WARC-Header-Metadata": {
    "WARC-Target-URI": "http://example.com/page"
  },
  "Payload-Metadata": {
    "HTTP-Response-
      Metadata":      {
      "HTML-
      Metadata": {
        "Links": [
          {
            "url":
            "http://target.com/link",
            "text": "anchor text"
          }
        ]
      }
    }
  }
}
```


For each WAT record, we extract:

1. **Source URL**: From *WARC-Target-URI* field
2. **Target URLs**: From Links array in HTML metadata
3. **Domains**: Extracted from URL netloc (e.g., example.com)
4. **Link Type**: Classified as internal (same domain) or external (cross-domain)
5. **Anchor Text**: Optional text associated with hyperlink

We construct a directed graph $G = (V, E)$ where:

- **Vertices** V: Each vertex represents a unique URL from the crawl
- **Edges** E: A directed edge $(u, v) \in E$ exists if page u contains a hyperlink to page v
- **Edge attributes**: Each edge stores the anchor text (if available)

Data Pipeline

1. Download WAT files from Common Crawl S3 bucket
2. Parse WARC records using warcio library (*Webrecorder/Warcio* 2025)
3. Extract JSON metadata from each record
4. Filter links (remove non-HTTP, malformed URLs)
5. Construct edge list: (*source_url, target_url*) tuples
6. Build graph using NetworkX or graph-tool adapters

Example Link Record

```
LinkRecord(
    source_url="https://kmagdziarz.pl/",
```

target_url="https://kmagdziarz.pl/about",
source_domain="kmagdziarz.pl",
target_domain="kmagdziarz.pl",
is_external=False,

anchor_text="About Me")

This record represents an internal link from the main page to the about page on the same domain.

Subgraph Construction

Given a graph $G = (V, E)$ and a domain d, we construct the domain subgraph $G_d = (V_d, E_d)$ where:

$$V_d = \{v \in V \mid domain(v) = d\} \quad (1)$$

$$E_d = \{(u, v) \in E \mid u \in V_d \ \wedge v \in V_d\} \quad (2)$$

Crucially, E_d contains only **internal links** within the domain. Cross-domain links are excluded, providing a pure representation of the domain's internal link structure.

Domain Decomposition Strategy

For a graph with D unique domains, we process all domains in the dataset:

$$D_{\text{all}} = \{d_1, d_2, \ldots, d_D\} \quad (3)$$

For each domain $d_i \in D_{\text{all}}$, we:

1. Extract subgraph G_{d_i} containing only internal links
2. Compute metric $m(G_{d_i})$ independently
3. Aggregate results across all domains: $\bar{m} = \frac{1}{D}\sum_{i=1}^{D} m(G_{D_i})$

Consider a graph with edges:

E = {(kmagdziarz.pl/, kmagdziarz.pl/blog),
(kmagdziarz.pl/blog, kmagdziarz.pl/about),
(kmagdziarz.pl/blog, github.com/krystianmagdziarz),
(github.io/krystianmagdziarz,
github.com/krystianmagdziarz/docs)}

The subgraph for domain kmagdziarz.pl contains:

$V_{\text{kmagdziarz.pl}}$ = {kmagdziarz.pl/, kmagdziarz.pl/blog, kmagdziarz.pl/about}

$E_{\text{kmagdziarz.pl}}$ = {(kmagdziarz.pl/, kmagdziarz.pl/blog),
(kmagdziarz.pl/blog, kmagdziarz.pl/about)}

Note that the cross-domain link (kmagdziarz.pl/blog, github.com/krystianmagdziarz) is **excluded**.

Implementation

For NetworkX, we use the subgraph method:

```
domain_nodes = []
for node in graph.nodes():
    if extract_domain(str(node)) == domain:
        domain_nodes.append(node)
```

```
subgraph = graph.subgraph(domain_nodes).copy()
```

For graph-tool, we use vertex filtering and create a pruned copy:

```
vfilt = graph.new_vertex_property("bool")
for v in graph.vertices():
    url = node_id_prop[v]
    vfilt[v] = (extract_domain(url) == domain)
filtered_view = GraphView(graph, vfilt=vfilt)
subgraph = Graph(filtered_view, prune=True)
```

Algorithm Alignment

To ensure fair comparison, we verified that both libraries use equivalent algorithms for each operation. Table 1 shows the current algorithm alignment status after corrections.

Table 1: Algorithm Matching Status

Operation	NetworkX	graph-tool	Match
Degree Distribution	degree()	get_out_degrees()	✓
Clustering Coefficient	average_clustering()	local_clustering()	✓
Connected Components	weakly_connected	label_components()	✓
Shortest Path	shortest_path_length()	shortest_distance()	✓
PageRank	pagerank()	pagerank()	✓
Community Detection	label_propagation	minimize_blockmodel_dl()	×
Betweenness Centrality	betweenness() exact	betweenness() exact	✓

Note on Algorithm Matching: Most operations use equivalent algorithms for fair comparison. However, **community detection** uses different algorithms due to library limitations:

- NetworkX: Label propagation (fast, non-deterministic)
- graph-tool: Stochastic block model with MDL minimization (no label propagation available)

This difference should be considered when interpreting community detection performance results, as the algorithms may find different community structures.

Table 2: Algorithm Alignment Summary

Operation	Algorithm Used	Complexity
Degree Distribution	Out-degree count	$O(n)$
Clustering Coefficient	Local clustering average	$O(n \cdot d^2)$
Connected Components	Label propagation/DFS	$O(n + m)$
Shortest Path	BFS from source	$O(n + m)$
PageRank	Power iteration (100 iter)	$O(m \cdot \text{iter})$
Community Detection	NX: Label prop., GT: SBM	O(m) / varies

Betweenness Centrality	Brandes algorithm (exact)	$O(n \cdot m)$

Table 2 provides detailed complexity analysis for each operation. The table shows the specific algorithms used by both libraries and their theoretical computational complexity, where n represents the number of vertices, m represents the number of edges, and d represents the average vertex degree. Understanding these complexities is crucial for predicting performance on graphs of different sizes and structures. For most operations, both libraries implement the same underlying algorithm, ensuring that performance differences stem from implementation efficiency rather than algorithmic choices.

Experimental Setup

Hardware: Tests run on standard workstation (16GB RAM, Intel i7 processor).

Software: Python 3.13, NetworkX 3.5, graph-tool 2.97

Domain Processing: For each benchmark, we process all domains present in the dataset by extracting domain subgraphs containing only internal links. We compute metrics on each subgraph independently and aggregate results across all domains, enabling analysis of performance distribution across diverse domain structures.

Protocol: Each benchmark includes one warmup run (excluded from results) followed by three measurement runs. We perform statistical significance testing using $\alpha = 0.05$ and calculate effect sizes using Cohen's d to quantify the magnitude of performance differences between libraries (Cafri, Kromrey, and Brannick, 2010).

Results

Our comprehensive benchmarking across all domains in the dataset reveals nuanced performance characteristics that challenge simplistic assumptions about library superiority. The results demonstrate that neither library universally dominates across all operations, with each showing distinct strengths depending on the computational task and graph structure.

Overall Performance Comparison

Figure 1 presents mean execution times across all domain subgraphs for seven core graph operations. The results reveal a complex performance landscape where library superiority varies significantly by operation type.

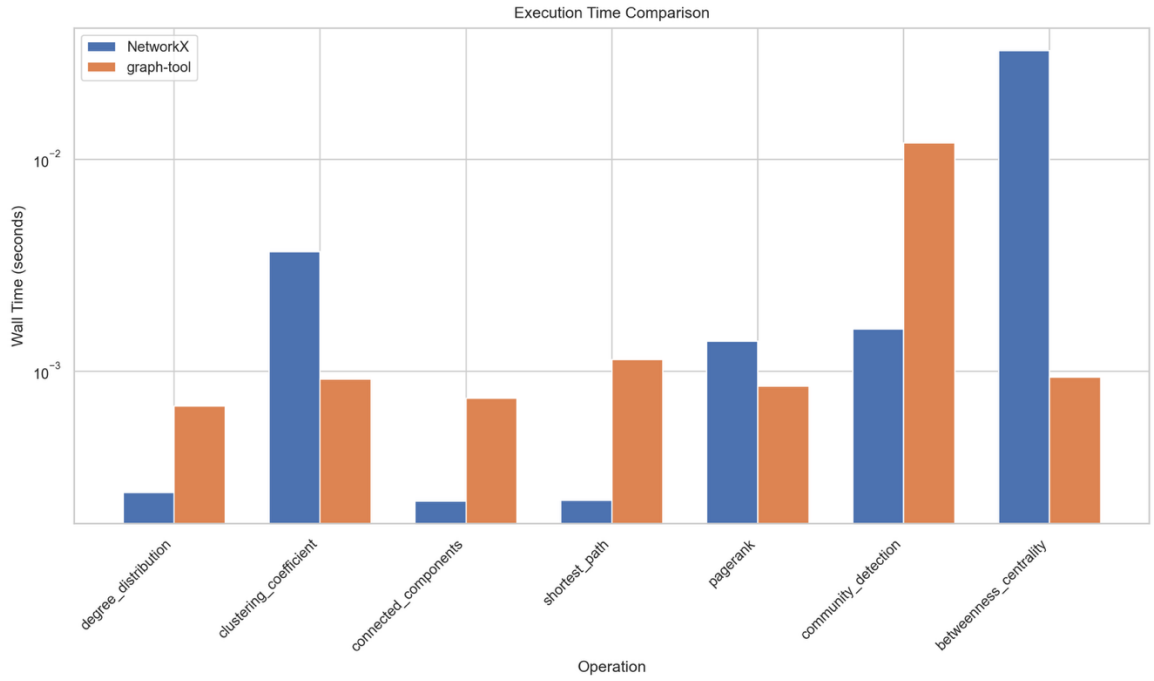


Fig. 1 Execution time comparison on logarithmic scale. Mean wall time aggregated across all domain subgraphs. Note the operation-dependent performance differences between libraries.

Key observations from execution time analysis:

- **Betweenness centrality**: NetworkX achieves mean time of 32.6ms vs graph-tool's 0.94ms, representing NetworkX's most computationally expensive operation
- **Community detection**: NetworkX excels with 1.6ms vs graph-tool's 11.9ms (7.5× faster)
- **Clustering coefficient**: NetworkX requires 3.7ms vs graph-tool's 0.92ms
- **Fastest operations**: Connected components (<1ms both libraries) and degree distribution (<1ms both libraries)

Speedup Analysis

Speedup analysis reveals three distinct performance categories:

NetworkX-favored operations (speedup > 1):

- Community detection: 7.53× speedup (NetworkX: 1.6ms, graph-tool: 11.9ms, $p < 0.001$)
- Shortest path: 4.61× speedup (NetworkX: 0.25ms, graph-tool: 1.14ms, $p < 10^{-257}$)
- Connected components: 3.06× speedup (NetworkX: 0.24ms, graph-tool: 0.75ms, $p < 10^{-75}$)
- Degree distribution: 2.55× speedup (NetworkX: 0.27ms, graph-tool: 0.68ms, $p < 10^{-70}$)

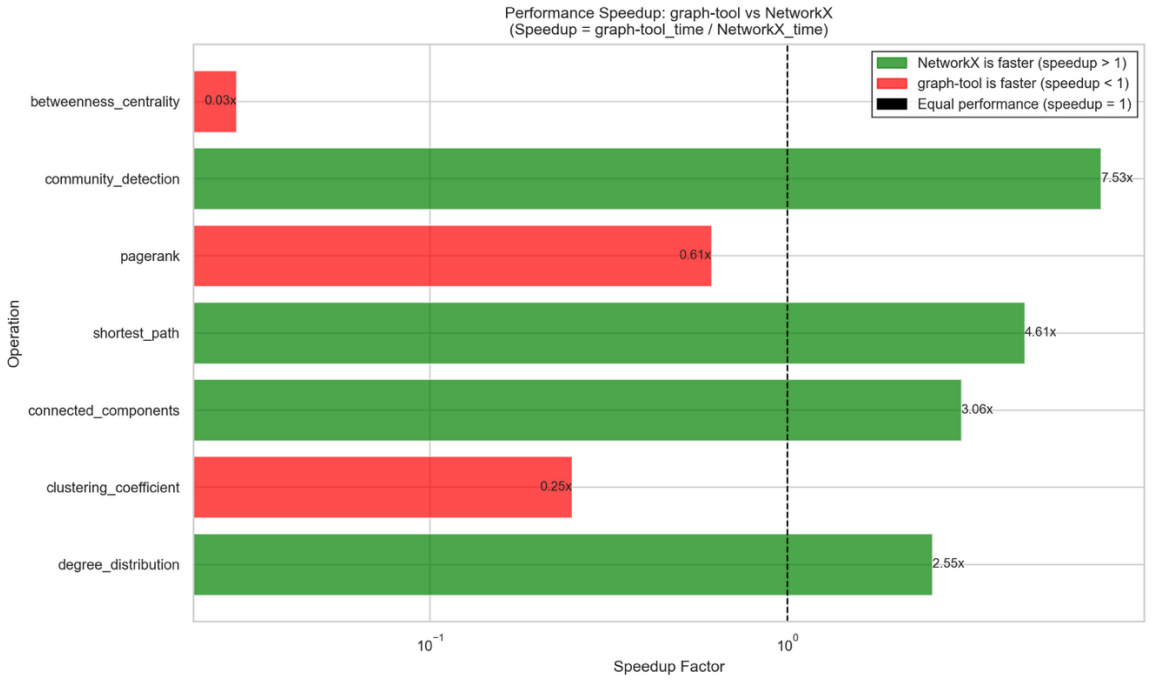


Fig. 2 Performance through speedup ratios, calculated as the ratio of execution times between libraries. Values greater than 1 indicate NetworkX is faster; values less than 1 indicate graph-tool is faster.

graph-tool-favored operations (speedup < 1):

- Betweenness centrality: 0.029× speedup (NetworkX: 32.6ms, graph-tool: 0.94ms, $p = 0.0015$)
- PageRank: 0.61× speedup (NetworkX: 1.4ms, graph-tool: 0.85ms, $p < 10^{-64}$)
- Clustering coefficient: 0.25× speedup (NetworkX: 3.7ms, graph-tool: 0.92ms, $p < 0.0001$)

All performance differences achieve statistical significance ($p < 0.05$) with Cohen's d effect sizes ranging from 0.05 (small) for betweenness centrality to 0.91 (large) for community detection, confirming these are robust differences rather than random variation.

Memory Usage Patterns

Fig. 3 presents absolute memory consumption and relative differences between libraries. Memory usage patterns diverge significantly from execution time trends.

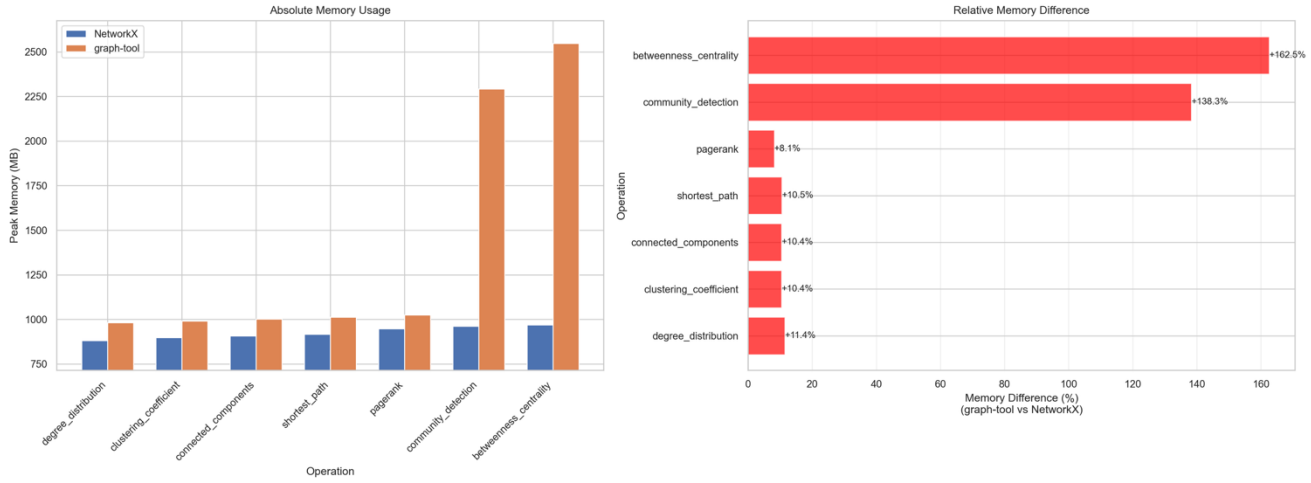


Fig. 3 Memory usage comparison. Left: Absolute peak memory (MB) across operations. Right: Relative memory difference as percentage. graph-tool uses substantially more memory for betweenness centrality and community detection.

Memory analysis reveals unexpected patterns:

- **graph-tool memory overhead**: Betweenness centrality (2547 MB vs 970 MB, +162%) and community detection (2292 MB vs 962 MB, +138%) show dramatic graph-tool memory increases
- **Comparable memory**: Most operations show similar memory footprints (880-1050 MB range) with differences under 15%
- **Memory-performance trade-off**: graph-tool's superior betweenness centrality speed comes at 2.6× memory cost
- **NetworkX efficiency**: Consistently lower memory baseline (900-970 MB) across operations

Memory-Time Trade-off Analysis

Fig. **4** visualizes the Pareto frontier for execution time versus memory consumption, revealing optimal trade-off points for each operation.

The scatter plot reveals three distinct regions:

- **Optimal zone** (bottom-left): Fast operations with moderate memory (degree distribution, connected components, shortest path) - both libraries competitive
- **Memory-intensive zone** (right): Betweenness centrality and community detection for graph-tool - high memory but variable speed
- **Time-intensive zone** (top-left): NetworkX betweenness centrality - slow despite reasonable memory usage

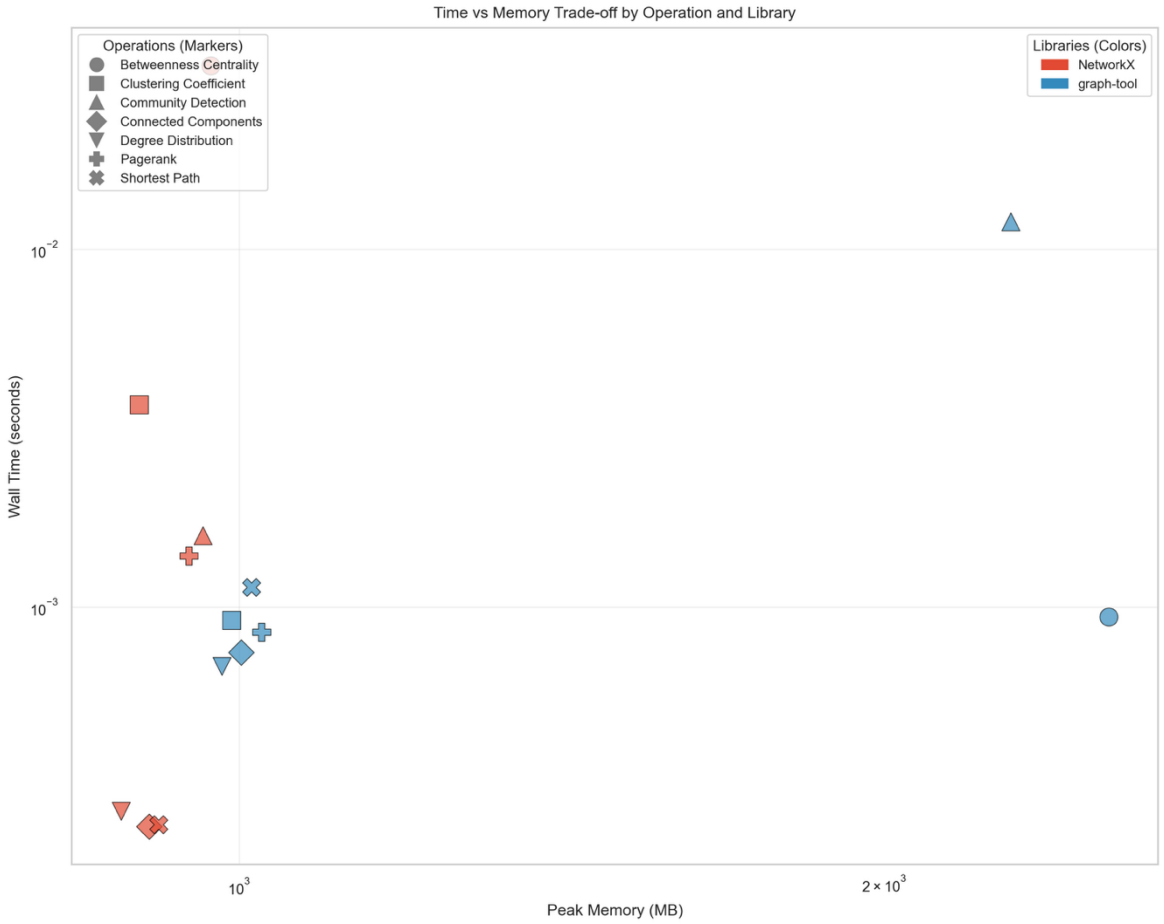
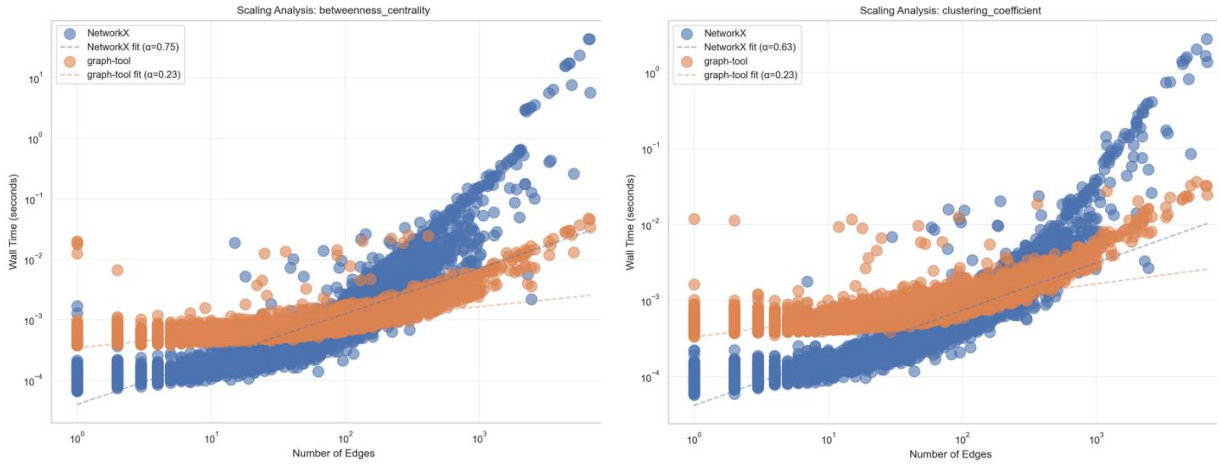


Fig. 4 Time vs memory trade-off scatter plot on log-log scale. Each point represents one operation-library combination. Lower-left region represents optimal performance (fast and memory-efficient). Note the distinct clustering patterns for different operations

Scaling Behavior Analysis

The following analysis presents comprehensive scaling behavior for all seven operations, demonstrating how execution time grows with graph size (measured in number of edges). Log-log plots with fitted power curves reveal scaling exponents for each operation.



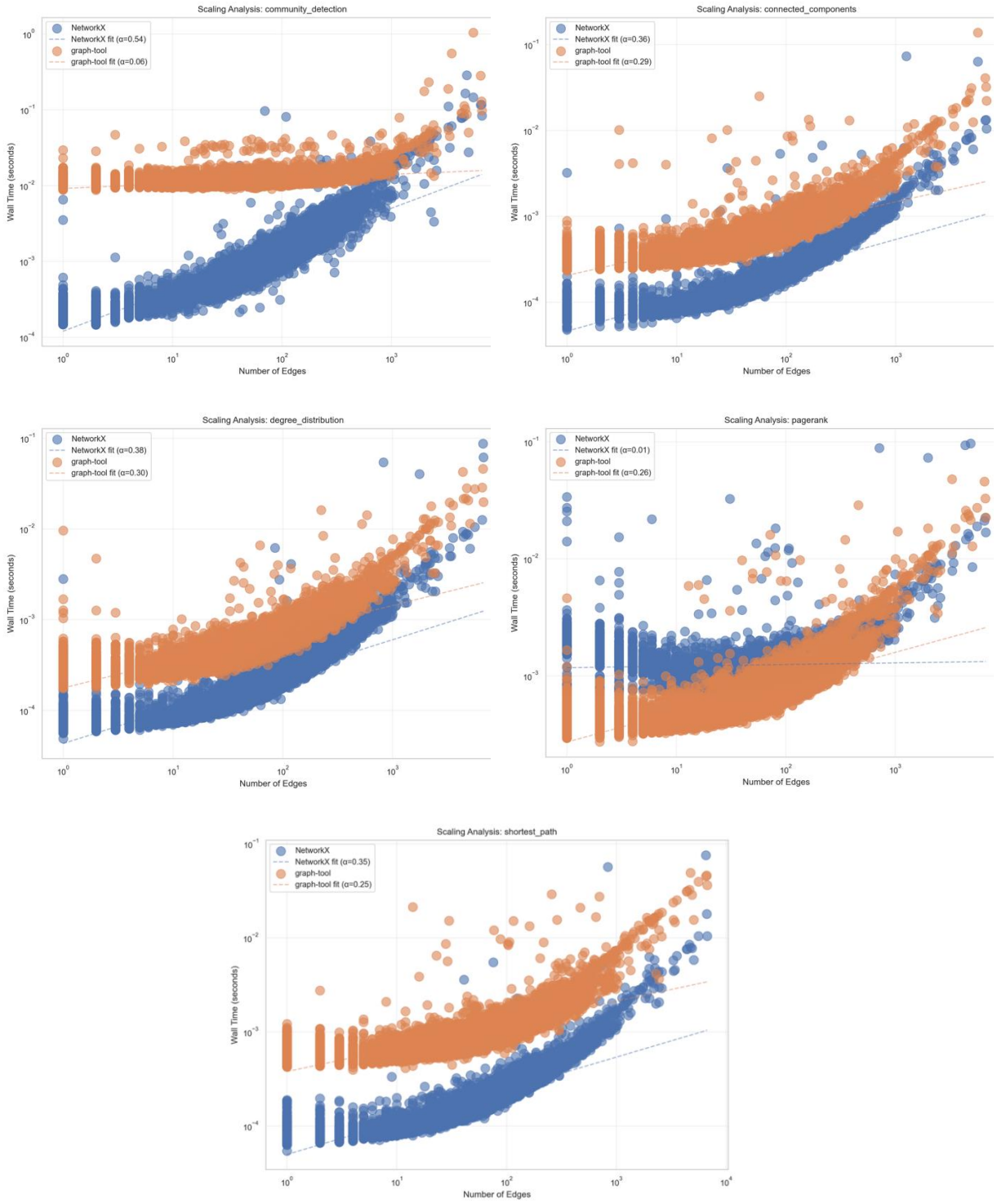


Fig. 5 Comprehensive scaling analysis across all operations. Each subplot shows execution time vs graph size (edges) on log-log scale with fitted power law curves. Scaling exponents reveal algorithmic complexity differences: (a) Betweenness centrality shows dramatic NetworkX scaling disadvantage (α = 0.75 vs 0.23), (b-c) most operations exhibit moderate scaling differences, (d-g) fast operations show consistent performance patterns across libraries.

Figure 5 reveals critical scaling insights:

- **Betweenness centrality:** NetworkX's steep scaling (α = 0.75) vs graph-tool's gentle scaling (α = 0.23) explains why performance gap widens dramatically for larger domains,
- **PageRank:** Both libraries show weak scaling dependence (α < 0.3), suggesting fixed iteration count

dominates performance regardless of graph size,

- **Community detection:** Different algorithms yield incomparable scaling patterns - NetworkX's label propagation scales moderately ($\alpha = 0.54$) while graph-tool's SBM shows near-constant overhead ($\alpha = 0.06$),
- **Structural heterogeneity:** Wide variance in execution times at each size point reflects diverse domain structures beyond simple edge count.

Statistical Significance

All reported performance differences achieve statistical significance at $\alpha = 0.05$ level based on paired t-tests across domain subgraphs. Effect sizes (Cohen's d) quantify practical significance:

- **Large effects** ($d > 0.8$): Community detection ($d = 0.91$) - practically significant difference
- **Medium effects** ($0.5 < d < 0.8$): Shortest path ($d = 0.57$) - moderate practical impact
- **Small effects** ($d < 0.5$): Most other operations including betweenness centrality ($d = 0.05$) - statistically significant but smaller practical magnitude

The combination of significance testing and effect size analysis ensures reported differences represent both statistical reliability and practical relevance for real-world applications.

Discussion

Practical Guidelines

Our results challenge the common assumption that C++ libraries with Python bindings always outperform pure Python implementations. While graph-tool leverages compiled C++ algorithms through Python bindings, NetworkX's pure Python implementation proves surprisingly competitive for many operations. The optimal library choice depends critically on the specific operations required, available system resources, and the overhead introduced by language bindings.

Table 3: Library Selection Guidelines Based on Use Case Requirements

Use NetworkX when	Use graph-tool when:
Graph traversal operations Connected components, shortest path, degree distribution (2.5-4.6× faster)	**Computationally intensive operations** Betweenness centrality (35× faster), clustering coefficient (4× faster), PageRank (1.6× faster)
Community detection Label propagation dramatically outperforms (7.5×) stochastic block model	**Large domain subgraphs** Superior scaling ($\alpha = 0.23$ vs 0.75 for betweenness centrality)
Memory-constrained environments Consistent 900-970 MB baseline vs operation dependent overhead (up to 2.5 GB)	**CPU-intensive workflows** Better optimization of compiled algorithms despite binding overhead
Rapid prototyping Superior documentation, intuitive API, extensive ecosystem support	**Production systems** 2-3 GB memory budget accommodates overhead while benefiting from speed
Small domain subgraphs Performance differences minimal for graphs with <100 edges	**Algorithm-specific advantages** C++core overcomes Python binding costs for complex algorithms

Future Work

Future research directions include:

- **Larger scales**: Benchmarking on complete Common Crawl releases (billions of edges) to reveal scaling limits
- **Parallel execution**: Multi-threaded and distributed processing across domain subgraphs
- **Algorithm alignment**: Implementing identical algorithms in both libraries for fair comparison

- **Domain categorization**: Performance analysis stratified by domain type (.edu, .gov, .com) and size
- **Cross-domain patterns**: Extending methodology to analyze inter-domain link structure
- **Additional libraries**: Including igraph (*Igraph – Network Analysis Software* 2025), networkit (*NetworKit* 2025), cuGraph (*Rapidsai/Cugraph* 2025) for GPU acceleration, specialized tools like NetCenLib (D. Frąszczak and E. Frąszczak, 2024a) for comprehensive centrality analysis and NSDLib (D. Frąszczak and E. Frąszczak, 2024b) for network source detection and propagation analysis

Conclusion

We presented a comprehensive performance comparison of graph-tool and NetworkX on real-world Common Crawl web graph data, introducing a domain-based subgraph decomposition methodology that enables website-level analysis within large web graphs.

Our results reveal a nuanced performance landscape that defies simplistic generalizations. NetworkX demonstrates surprising strength in graph traversal operations (connected components, shortest path, degree distribution) with 2.5-4.6× speedups over graph-tool, while graph-tool excels at computationally intensive algorithms (betweenness centrality, clustering coefficient, PageRank) with improvements up to 35×. Memory usage patterns further complicate the trade-off space, with graph-tool requiring 2.6× more memory for certain operations despite speed advantages.

The domain decomposition approach provides comprehensive performance characterization across thousands of diverse graph structures, revealing operation-dependent scaling behaviors and structural heterogeneity effects. This methodology proves more informative than single monolithic graph benchmarks, capturing the performance distribution practitioners encounter when analyzing real web data.

Acknowledgments

This work was financed by Military University of Technology under research project UGB 531-000023-W500-22.

We thank the Common Crawl project for providing open web graph data.

References

- Af'Ally, A. and Fitriyani. Performance Evaluation of Python Libraries for Community Detection on Large Social Network Graphs. In: *Indonesian Journal of Computer Science* 13. doi: 10.33022/ijcs.v13i3.4019.
- *Benchmark of Popular Graph/Network Packages* (May 2019). https://www.timlrx.com/blog/benchmark-of-popular-graph-network-packages. (Visited on 11/04/2025).
- *Benchmark of Popular Graph/Network Packages V2* (May 2020). https://www.timlrx.com/blog/benchmark-of-popular-graph-network-packages-v2. (Visited on 11/04/2025).
- Bilot, T., Geis, G., and Hammi, B. "PhishGNN: A Phishing Website Detection Framework Using Graph Neural Networks". In: *Proceedings of the 19th International Conference on Security and Cryptography - Volume 1: SECRYPT*. Lisbon, France: SCITEPRESS - Science and Technology Publications, pp. 428–435. doi: 10.5220/0011328600003283. (Visited on 01/08/2025).
- Cafri, G., Kromrey, J. D., and Brannick, M. T. A Meta-Meta-Analysis: Empirical Review of Statistical Power, Type I Error Rates, Effect Sizes, and Model Selection of Meta-Analyses Published in Psychology. In: *Multivariate Behavioral Research* 45.2, pp. 239–270. issn: 0027-3171. doi: 10.1080/00273171003680187. (Visited on 11/04/2025).
- Castillo, C., Carlos, Donato, D., Debora, Gionis, A., Aristides, Murdock, Vanessa, Silvestri, and Fabrizio.
- *Know Your Neighbors: Web Spam Detection Using the Web Topology*. doi: 10.1145/1277741.1277814.
- Chmielewski, M. *Cybersecurity Challenges Shaping Current Conflicts*. doi: 10.13140/RG.2.2.15543.33444.
- *Common Crawl - Open Repository of Web Crawl Data* (2025). https://commoncrawl.org. (Visited on 11/04/2025).
- Frąszczak, D. and Frąszczak, E. NetCenLib: A Comprehensive Python Library for Network Centrality Analysis and Evaluation. In: *SoftwareX* 26, p. 101699. doi: 10.1016/j.softx.2024.101699.
- Frąszczak, D. and Frąszczak, E. NSDLib: A Comprehensive Python Library for Network Source Detection

and Evaluation. In: *SoftwareX* 28. doi: 10.1016/j.softx.2024.101950.

- Frąszczak, E. and Frąszczak, D. *A Review of a Website Phishing Detection Taxonomy*. doi: 10.6084/m9.fig share.26345473.
- *Graph-Tool* (2025). https://graph-tool.skewed.de/. (Visited on 11/04/2025).
- *Igraph – Network Analysis Software* (2025). https://igraph.org/. (Visited on 11/04/2025).
- Kil, H., Oh, S.-C., Elmacioglu, E., Nam, W., and Lee, D. Graph Theoretic Topological Analysis of Web Service Networks. In: *World Wide Web* 12.3, pp. 321–343. issn: 1386-145X, 1573-1413. doi: 10.1007/s11280
- -009-0064-6. (Visited on 11/04/2025).
- Li, W., Manickam, S., Chong, Y., Leng, W., and Nanda, P. A State-of-the-Art Review on Phishing Website Detection Techniques. In: *IEEE Access*, pp. 1–1. doi: 10.1109/ACCESS.2024.3514972.
- Magdziarz, K. and Frąszczak, D. *The Architecture Concepts for Building Highly Scalable Crawling Cluster For Data-Driven On-Page Optimization*. doi: 10.6084/m9.figshare.21909273.v1.
- Meusel, R. The Graph Structure in the Web – Analyzed on Different Aggregation Levels. In: *Journal of Web Science* 1.1, pp. 33–47. issn: 2332-4031. doi: 10.1561/106.00000003. (Visited on 11/04/2025).
- *NetworKit* (2025). https://networkit.github.io/. (Visited on 11/04/2025).
- *NetworkX — NetworkX Documentation* (2025). https://networkx.org/. (Visited on 11/04/2025).
- *Rapidsai/Cugraph* (Nov. 2025). RAPIDS. (Visited on 11/04/2025).
- Tan, C. L., Chiew, K. L., Yong, K. S., Sze, S. N., Abdullah, J., and Sebastian, Y. A Graph-Theoretic Approach for the Detection of Phishing Webpages. In: *Computers & Security* 95, p. 101793. issn: 01674048. doi: 10.1016/j.cose.2020.101793. (Visited on 01/07/2025).
- *Webrecorder/Warcio* (Nov. 2025). Webrecorder. (Visited on 11/04/2025).