Dynamic Parallel Maintaince of Strongly Connected Components in MPI

A Report submitted in partial fulfillment for the Degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Soham Tripathy*

Supervised by Professor. Rupesh Nasre

Submitted April 28, 2024

*Student ID: CS20B073, Email: cs20b073@smail.iitm.ac.in

Abstract

This report explores the criticality of maintaining strongly connected components (SCCs) amidst dynamic changes in graph structures, emphasizing the efficiency of preserving SCCs during edge deletions and additions. Understanding the significance of SCCs in various applications, it becomes very challenging to recompute SCCs in response to changes in the graph structure. This requires efficient algorithms and optimization techniques to minimize the computational overhead and ensure real-time updates. Building upon prior research, we worked on extending an optimal algorithm tailored for this purpose. Each facet of the algorithm deals with various cases of graph updates, which is elucidated with illustrative examples and accompanying pseudo code.

Leveraging MPI for distributed parallel computation, we implement our algorithm, addressing encountered challenges and intricacies of the implementation process to facilitate comprehension of the code. Detailed explanations of the design implementation, including the SCC tree construction, decremental and incremental maintenance, are provided. Rigorous testing on standard and specialized graphs, augmented with dynamic updates, is conducted, juxtaposing the performance against static SCC algorithms. Results that are depicted through graphs and tables, are analyzed in the conclusion, offering insights and proposing avenues for future enhancements. Additionally, the appendix elucidates the functionality and testing procedures of the implemented library, ensuring comprehensive understanding and usability.

Table of Contents

Conventions and Notation

i. A convention you use here, e.g. Unless otherwise stated we use units such that $c = 1$.

List of Figures

List of Tables

Acknowledgements

I would like to express my heartfelt gratitude to *Professor Rupesh Nasre*, Department of Computer Science and Engineering, Indian Institute of Technology, Madras for his invaluable guidance, unwavering support, and insightful feedback throughout the duration of this project. His expertise and mentorship have been instrumental in shaping the direction of this work.

Additionally, I extend my sincere appreciation to the M.Tech students, *Barenya Kumar Nandy* and *Anurag Sao*, whose valuable insights, references, and constant assistance have greatly enriched this endeavor.

I would also like to thank the Indian Institute of Technology, Madras, for providing the necessary resources and infrastructure for the successful completion of this project.

Soham Tripathy

Main Content

1 Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. [1, 1]

2 Literature Review

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

3 Theoretical Methodology

In this part, we elaborate on and enhance the algorithm proposed in [2] for maintaining strongly connected components (SCCs) under a sequence of updates. The algorithm is designed to accommodate n vertices and m edges as input, proficiently manages the following operations:

- Query(u, v): Verifies whether both u and v belong to the same SCC.
- Delete(u, v): Removes the edge from u to v.
- Add(u, v): Introduces the edge from u to v.

This algorithm achieves its objectives through the creation and continuous maintenance of a specialized data structure known as the SCC Tree, which is further explained in the subsequent section. Additionally, it employs an SCC mapping array to facilitate queries in constant time *O(1)*.

The algorithm initiates with an initialization stage, wherein it constructs the proposed SCC tree and populates the SCC mapping array based on the identified strongly connected components within the graph. Subsequently, the process of constructing and maintaining these pivotal data structures under the delete and add updates is elaborated in the following sections, accompanied by their respective pseudo codes.

3.1 Data Structures

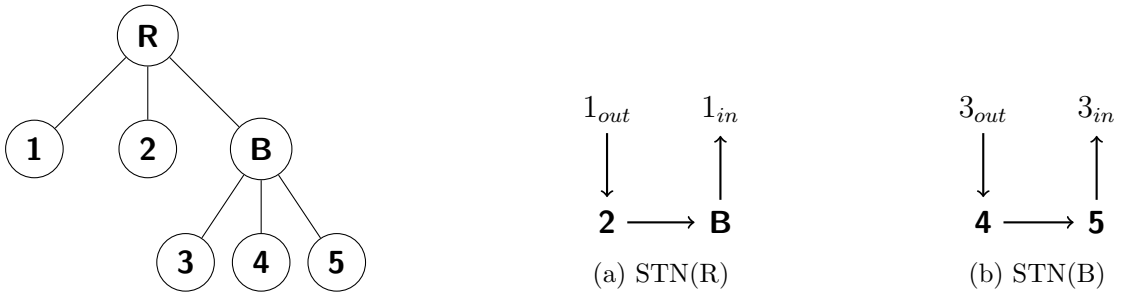
In this section, we delve into a comprehensive exploration of the specialized data structures crucial to the functionality and efficiency of the algorithm. Through this detailed examination, we aim to provide clarity and insight into the design principles, operational mechanisms, and computational complexities of these structures.

3.1.1 SCC Tree

The SCC tree serves as a vital component in maintaining the internal connectivity of vertices within the strongly connected components of graph G. For each SCC identified in the graph, a corresponding SCC tree is constructed.

The node in the SCC tree corresponding to the label R encapsulates a tuple that can be represented as $STN(R) = (V, E)$, where V represents set of vertices and E represents set of edges connecting them. This encapsulation enables the SCC tree to maintain the connectivity of the vertices that are a part of the SCC labeled R, also facilitating efficient traversal and update operations.

Each vertex $v \in V$ in the STN of R is a label uniquely associated with an SCC tree. This tree is a subtree of the SCC tree represented by the label R. The SCC tree for a graph containing one vertex v has only a single STN represented as $STN(v) = (\{v\}, \emptyset)$.



(a) STN(R) (b) STN(B)

Figure 1: Example SCC Tree and its corresponding STNs

The SCC tree shown in Fig.1, can be represented by a collection of STNs as follows:

- $STN(R) = (\{1, 2, B\}, \{(1, 2), (2, B), (B, 1)\})$
- $STN(B) = (\{3, 4, 5\}, \{(3, 4), (4, 5), (5, 3)\})$

- $STN(v) = (\{v\}, \emptyset) \, \forall v \in \{1, 2, 3, 4, 5\}$

The SCC tree represented by label B contains the STNs of $\{B, 3, 4, 5\}$, which form a subtree of the SCC tree represented by label R, similarly, the SCC tree represented by labels $\{1, 2\}$ also form R's subtree.

3.1.2 SCC Mapping Array

In graph G, each vertex v is inherently associated with a strongly connected component (SCC), denoted by its corresponding SCC label. The SCC mapping array effectively captures this relationship between vertices and their respective SCC labels.

Suppose vertex v in graph G belongs to the strongly connected component labeled as R. In that case, we express this association using the SCC mapping array notation as $SM_G(v) = R$. This signifies that vertex v is mapped to the SCC labeled as R within the context of the graph G.

3.2 Definitions

In this section, we establish key definitions that form the foundation of the algorithms presented subsequently. These definitions are instrumental in understanding the intricacies of the algorithms and their associated data structures.

3.2.1 ***FIND_SCC*(G)**

3.2.2 ***CONDENSE*(G)**

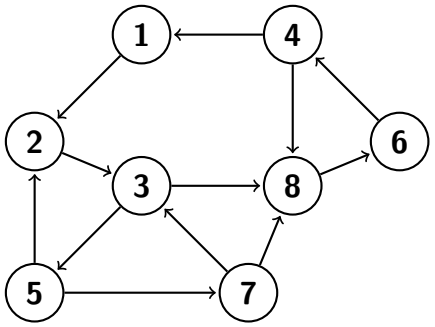
3.2.3 ***SPLIT*(G)**

3.2.4 ***MERGE*(G)**

3.2.5 ***UNREACHABLE*(G)**

3.3 Constructing SCC Tree

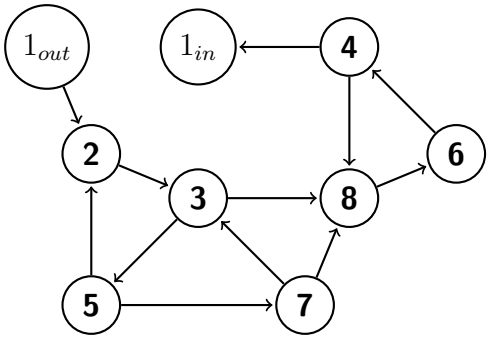
We will look at the construction of the SCC tree for the graph shown in Fig.2a by using the algorithm described in the section §3.1.1. We start by finding all the SCCs of the graph and then construct the SCC-Tree for each SCC in it. This is also used to fill the SCC mapping later used to query the SCC of a node in constant time.
Let the example graph be **G**, by running FIND_SCC(**G**) we get SCC(R)



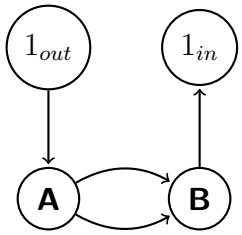
(a) Graph 1

(b) condensed graph 1

Figure 2: Graph 1 and its condensed graph

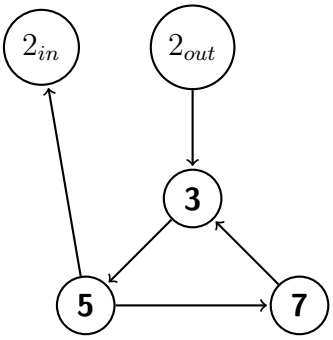


(a) SCC(R) split on 1

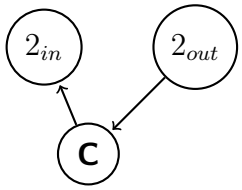


(b) condensed graph

Figure 3: SCC(R) split on 1 and its condensed graph

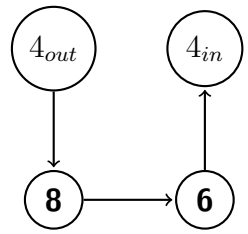


(a) SCC(A) split on 2

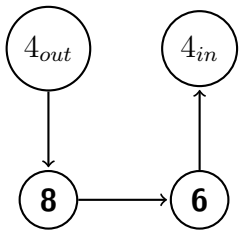


(b) condensed graph

Figure 4: SCC(A) split on 2 and its condensed graph

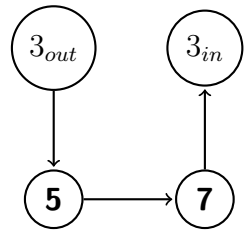


(a) SCC(B) split on 4

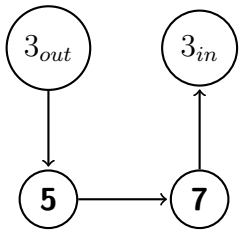


(b) condensed graph

Figure 5: SCC(B) split on 4 and its condensed graph



(a) SCC(C) split on 3



(b) condensed graph

Figure 6: SCC(C) split on 3 and its condensed graph

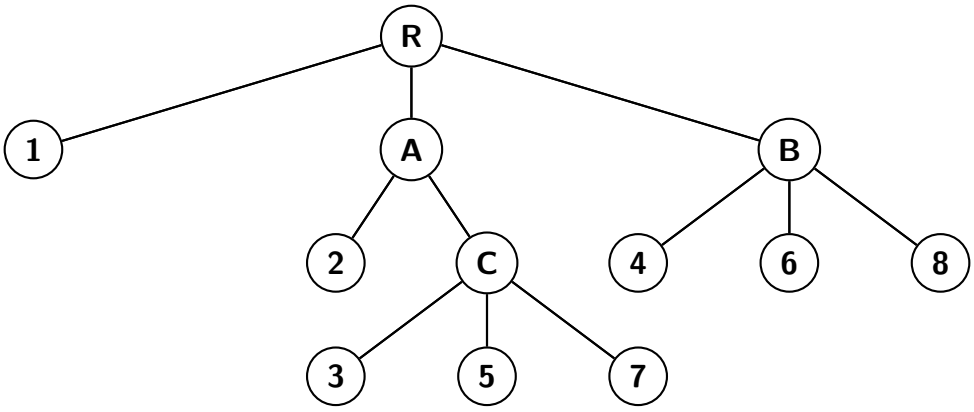
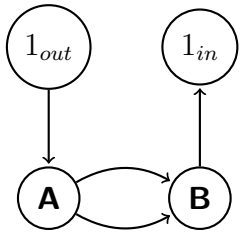


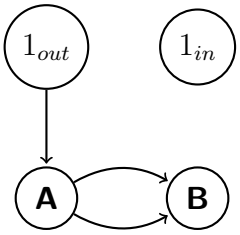
Figure 7: SCC Tree of Graph 2a

3.4 Decremental Maintaince of SCC Tree

3.4.1 Deleting Edge from the Root Node

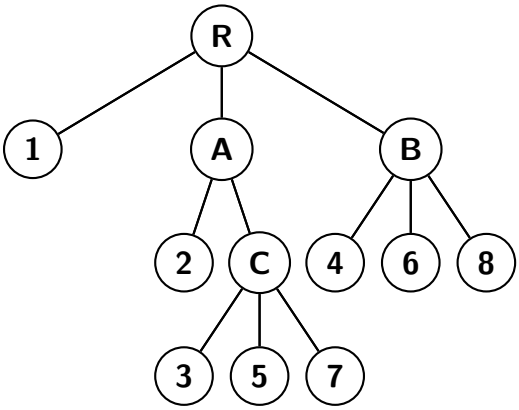


(a) Graph in SCC Tree Node R

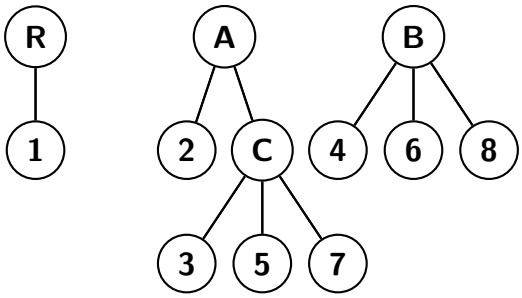


(b) Graph after deleting edge 4 to 1

Figure 8: Graph in SCC Tree Node R after deleting edge 4 to 1

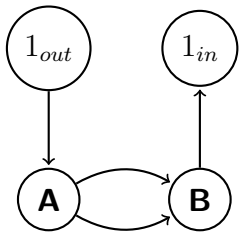


(a) SCC Tree of Graph 2a

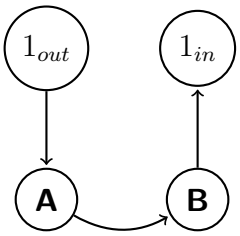


(b) SCC Tree of Graph 2a after updates

Figure 9: SCC Tree updates are propogated, deleting edge 4 to 1



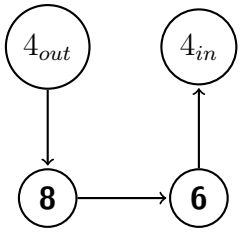
(a) Graph in SCC Tree Node R



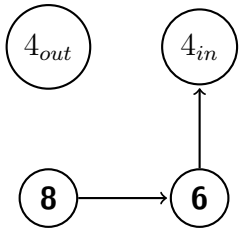
(b) Graph after deleting edge 3 to 8

Figure 10: Graph in SCC Tree Node R after deleting edge 3 to 8

3.4.2 Deleting Edge from an Internal Node

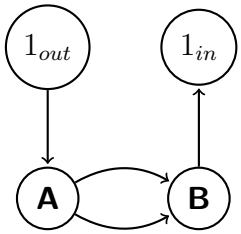


(a) Graph in Tree Node B

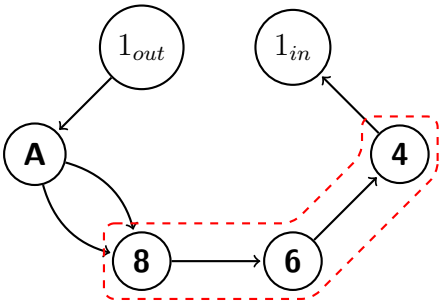


(b) Graph after deleting edge 4 to 8

Figure 11: Graph in Tree Node B after deleting edge 4 to 8

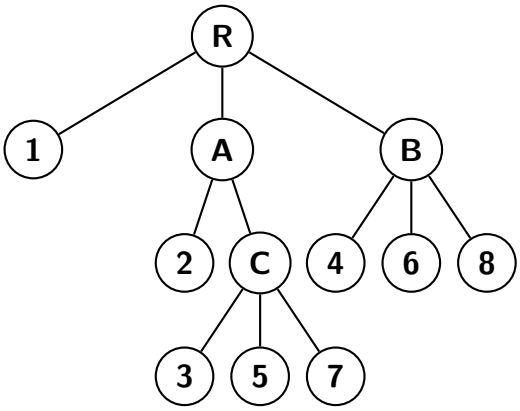


(a) Graph in SCC Tree Node R

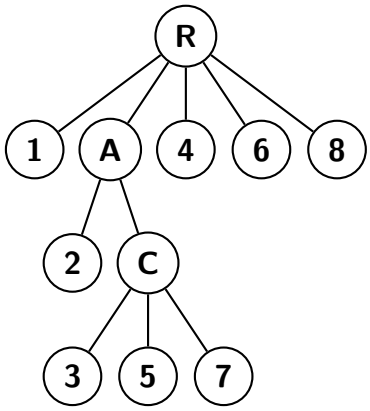


(b) After Exposure

Figure 12: Graph in SCC Tree Node R after unreachable nodes and its corresponding edges are exposed



(a) SCC Tree of Graph 2a



(b) SCC Tree of Graph 2a after updates

Figure 13: Tree updates are propogated, deleting edge 4 to 8

Algorithm 1: Example Algorithm

Data: Input data
Result: Output result
initialization;
while *termination condition is not met* **do**
    perform action;
    update state;
end

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

Listing 1: Example C++ code

3.5 Incremental Maintaince of SCC Tree

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

4 Practical Implementation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

4.1 Data Structures

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

4.2 Workload Distribution in Parallel

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

4.3 Update Cache Optimizations

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Table 1: Timings with/without Cache Optimizations

Updates(%)	With Caching	Without Caching
0	3,072,441	234,370,166
1	58,655,849	261,321,071
5	3,370,462	93,373,056
10	4,847,571	68,993,773
15	1,632,803	30,622,564
20	21,297,772	265,025,809
30	16,777,216	87,654,320
50	10,000,000	80,000,000

4.4 Challenges in Implementation with MPI

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

5 Results

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Table 2: Graph Details

Graph	Number of Vertices	Number of Edges
cleancom-orkutud-SCC	3,072,441	234,370,166
clean-soc-sinaweibo-SCC	58,655,849	261,321,071
cleanWikipedia-SCC	3,370,462	93,373,056
livjournal-SCC	4,847,571	68,993,773
clean-soc-pokec-relationships-SCC	1,632,803	30,622,564
clean-soc-twitter-SCC	21,297,772	265,025,809
rmat876-SCC	16,777,216	87,654,320
u10m_80m-SCC	10,000,000	80,000,000

Table 3: Timings for Graph cleancom-orkutud-SCC

Update(%)	Decremental		Incremental		Both	
w.r.t. edges	Static	Dynamic	Static	Dynamic	Static	Dynamic
0	1.0	1.1	1.2	1.3	1.4	1.5
1	2.3	2.4	2.5	2.6	2.7	2.8
5	2.9	3.0	3.1	3.2	3.3	3.4
10	3.5	3.6	3.7	3.8	3.9	4.0
15	4.1	4.2	4.3	4.4	4.5	4.6
20	4.1	4.2	4.3	4.4	4.5	4.6
30	4.7	4.8	4.9	5.0	5.1	5.2

Table 4: Timings for Graph clean-soc-sinaweibo-SCC

Update(%)	Decremental		Incremental		Both	
w.r.t. edges	Static	Dynamic	Static	Dynamic	Static	Dynamic
0	1.0	1.1	1.2	1.3	1.4	1.5
1	2.3	2.4	2.5	2.6	2.7	2.8
5	2.9	3.0	3.1	3.2	3.3	3.4
10	3.5	3.6	3.7	3.8	3.9	4.0
15	4.1	4.2	4.3	4.4	4.5	4.6
20	4.1	4.2	4.3	4.4	4.5	4.6
30	4.7	4.8	4.9	5.0	5.1	5.2

Table 5: Timings for Graph `cleanWikipedia-SCC`

Update(%)	**Decremental**		**Incremental**		**Both**	
w.r.t. edges	Static	Dynamic	Static	Dynamic	Static	Dynamic
0	1.0	1.1	1.2	1.3	1.4	1.5
1	2.3	2.4	2.5	2.6	2.7	2.8
5	2.9	3.0	3.1	3.2	3.3	3.4
10	3.5	3.6	3.7	3.8	3.9	4.0
15	4.1	4.2	4.3	4.4	4.5	4.6
20	4.1	4.2	4.3	4.4	4.5	4.6
30	4.7	4.8	4.9	5.0	5.1	5.2

Table 6: Timings for Graph `livjournal-SCC`

Update(%)	**Decremental**		**Incremental**		**Both**	
w.r.t. edges	Static	Dynamic	Static	Dynamic	Static	Dynamic
0	1.0	1.1	1.2	1.3	1.4	1.5
1	2.3	2.4	2.5	2.6	2.7	2.8
5	2.9	3.0	3.1	3.2	3.3	3.4
10	3.5	3.6	3.7	3.8	3.9	4.0
15	4.1	4.2	4.3	4.4	4.5	4.6
20	4.1	4.2	4.3	4.4	4.5	4.6
30	4.7	4.8	4.9	5.0	5.1	5.2

Table 7: Timings for `clean-soc-pokec-relationships-SCC`

Update(%)	**Decremental**		**Incremental**		**Both**	
w.r.t. edges	Static	Dynamic	Static	Dynamic	Static	Dynamic
0	1.0	1.1	1.2	1.3	1.4	1.5
1	2.3	2.4	2.5	2.6	2.7	2.8
5	2.9	3.0	3.1	3.2	3.3	3.4
10	3.5	3.6	3.7	3.8	3.9	4.0
15	4.1	4.2	4.3	4.4	4.5	4.6
20	4.1	4.2	4.3	4.4	4.5	4.6
30	4.7	4.8	4.9	5.0	5.1	5.2

Table 8: Timings for Graph `clean-soc-twitter-SCC`

Update(%)	**Decremental**		**Incremental**		**Both**	
w.r.t. edges	Static	Dynamic	Static	Dynamic	Static	Dynamic
0	1.0	1.1	1.2	1.3	1.4	1.5
1	2.3	2.4	2.5	2.6	2.7	2.8
5	2.9	3.0	3.1	3.2	3.3	3.4
10	3.5	3.6	3.7	3.8	3.9	4.0
15	4.1	4.2	4.3	4.4	4.5	4.6
20	4.1	4.2	4.3	4.4	4.5	4.6
30	4.7	4.8	4.9	5.0	5.1	5.2

Table 9: Timings for Graph `rmat876-SCC`

Update(%)	**Decremental**		**Incremental**		**Both**	
w.r.t. edges	Static	Dynamic	Static	Dynamic	Static	Dynamic
0	1.0	1.1	1.2	1.3	1.4	1.5
1	2.3	2.4	2.5	2.6	2.7	2.8
5	2.9	3.0	3.1	3.2	3.3	3.4
10	3.5	3.6	3.7	3.8	3.9	4.0
15	4.1	4.2	4.3	4.4	4.5	4.6
20	4.1	4.2	4.3	4.4	4.5	4.6
30	4.7	4.8	4.9	5.0	5.1	5.2

Table 10: Timings for Graph `u10m_80m-SCC`

Update(%)	**Decremental**		**Incremental**		**Both**	
w.r.t. edges	Static	Dynamic	Static	Dynamic	Static	Dynamic
0	1.0	1.1	1.2	1.3	1.4	1.5
1	2.3	2.4	2.5	2.6	2.7	2.8
5	2.9	3.0	3.1	3.2	3.3	3.4
10	3.5	3.6	3.7	3.8	3.9	4.0
15	4.1	4.2	4.3	4.4	4.5	4.6
20	4.1	4.2	4.3	4.4	4.5	4.6
30	4.7	4.8	4.9	5.0	5.1	5.2

6 Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede.

Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Appendices

A Example 1

B Example 2

Bibliography

[1] Author 1 and Author 2. A paper to reference. *Journal X*, 2024.

[2] Jakub Lacki. Improved deterministic algorithms for decremental reachability and strongly connected components. *ACM Trans. Algorithms, 9, 3, Article 27*, June 2013.