GSoC Proposal
May 2025 to August 2025

Organization: NumFOCUS (NetworkX)

Project: Adding embarrassingly parallel graph algorithms in nx-parallel

Personal Details

Name	Akshita Sure
Email	sureakshita23@gmail.com
Primary Language	English, Telugu
Time Zone	Indian Standard Time (UTC + 5:30)
GitHub	akshitasure12
LinkedIn	www.linkedin.com/in/akshitasure
Working Hours	11:00 AM - 12:00 AM IST (UTC + 5:30)

About Me

I'm a Computer Science Engineering student at Atal Bihari Vajpayee Indian Institute of Information Technology and Management (IIITM), Gwalior. I have a strong foundation in Data Structures and Algorithms, and I enjoy Competitive Programming.

Over the months, I've built hands-on experience with Python and Flask, developing a range of personal and academic projects. I'm also passionate about Deep Learning and enjoy experimenting with different algorithms and models. Recently, I've developed interest in open-source software and have started contributing to projects that align with my technical interests.

Apart from tech, I've been trained in music for over a decade and am deeply passionate about singing — it's my creative escape and something I hold close to my heart. I'm always eager to learn, build, and collaborate on meaningful projects that challenge me and expand my skill set.

Abstract

This proposal focuses on enhancing the `nx-parallel` backend of NetworkX by implementing parallel versions of embarrassingly parallel algorithms. The algorithms initially targeted for parallelization include: `harmonic_centrality()`, `triangles()`, `clustering_coefficient()`, `jaccard_coefficient()`, and `average_neighbor_degree()`. As I continue working, I also intend to identify and propose additional algorithms where parallelization can provide significant speed-ups. In addition to algorithm development, the project will address a current limitation related to inconsistent performance heatmaps and the existing timing script, as documented in nx-parallel#51. Updating and improving this script to generate consistent, reliable visual performance comparisons across different algorithms will also be a key deliverable of the project. I also aim to actively contribute by identifying and fixing any bugs encountered during development to help improve the robustness of the codebase overall.

Why NetworkX?

I've always been drawn to logical thinking which naturally led me to explore programming even before college. While exploring previous GSoC organizations, I noticed that many projects either did not align with my interests or were difficult to grasp at first glance.

When I discovered NetworkX, I immediately identified it as a project of my interest. It focused on graph theory and algorithms, areas in which I have a genuine interest and knowledge. I was also impressed by how clear and approachable the work with NetworkX was. After contributing, I realised that the community was very open and supportive, and this made the experience even more rewarding.

I am eager to continue working in this organization because it aligns perfectly with my interests, and I appreciate the welcoming nature of its community. Contributing to this project represents not only a good learning experience but also an opportunity to work on something that fascinates me.

My Contributions to NetworkX

#1 Contribution

Started | Closed
Nov 3, 2024 | Nov 4, 2024

Link: https://github.com/networkx/networkx/pull/7706#

This was my first contribution to the Networkx repository and also on GitHub as an open source contributor. I learnt a lot about FOSS, git, and version control using online resources while solving this issue.

Aditi Juneja (Schefflera-Arboricola) encouraged me to draft a Pull Request so that the work I had done could be reviewed better. Since this was my first time adding something directly to the NetworkX codebase, I familiarized myself with essential contributor tools like pre-commit linting and code formatting.

This contribution made the following changes:

- Resolved a package installation error related to `changelist==0.5`, which was no longer available in the specified channels.
- Added the missing installation command to the documentation because I noticed that the instructions for setting up pre-commit hooks were incomplete.

The pre-commit was already included in the `requirements/developer.txt`, making the previous explicit installation unnecessary. Additionally, with the approval of conda-forge/staged-recipes#28089, there was no longer the need to install changelist.

This made the contribution obsolete, and the PR was eventually closed.

I learned from this experience how crucial it is to fully comprehend the current infrastructure of a project before making any modifications. It emphasized the importance of thorough research and cooperation within large open-source communities like Networkx.

#2 Contribution

Started | Open
Dec 17, 2024 | N/A

Link: https://github.com/networkx/networkx/pull/7771

This was my second contribution to the library. The issue raised in #6873 highlights a problem with `complete_to_chordal_graph()`, specifically regarding its handling of maximum cardinality search (MCS) and the expected perfect elimination ordering (PEO). It returns an ordering where the graph is marked as chordal, but it does not generate a proper, perfect elimination ordering.

This contribution made the following changes:

- Added an efficient PEO algorithm to compute perfect elimination orderings for chordal graphs. The references for this implementation include:
  - https://www.ii.uib.no/~pinar/MCSM-r.pdf
  - https://en.wikipedia.org/wiki/Lexicographic_breadth-first_search
- Added the following tests to validate the functionality of the `perfect_elimination_ordering()` algorithm:

- **Test 1**: For a simple graph (e.g., a chain or path) or a graph with cycles, checks if the correct PEO order is returned.
- **Test 2**: Verifies that a non-chordal graph raises a `NetworkXError` with the message "Input graph is not chordal."
- **Test 3**: Ensures that an empty graph returns an empty list `[]`.
- **Test 4**: For a single-node graph, checks if the order `[1]` is returned.

The pull request for this enhancement is still open.

#3 Contribution

Started | Merged
Dec 18, 2024 | Dec 21, 2024

Link: https://github.com/networkx/networkx/pull/7773/

This was my third contribution to the library. This contribution was based on adding a note to the `contributing.rst` file, advising contributors to avoid using NumPy scalars as return types.

This is based on Issue #7349, which highlighted the need for consistent return types to maintain compatibility and prevent potential issues within the codebase.

Before making changes to the codebase :

- I explored the behavior of NumPy scalar types and how they differ from native Python scalars. This involved studying the NumPy Scalars documentation to understand their implications in mathematical operations and function outputs.

- To better understand the impact of NumPy scalars, I examined the NetworkX codebase to identify areas where using these scalars led to inconsistencies.

This contribution made the following changes to the project:

- Introduced a clear guideline in `contributing.rst` advising contributors to avoid returning NumPy scalars (`numpy.int64`, `numpy.float64`, etc.) to ensure better compatibility across the codebase.

- Included a practical code example demonstrating how to convert NumPy scalars to native Python types using `.item()`, making it easier for contributors to follow best practices.

This contribution was successfully reviewed and merged.

#4 Contribution

Started | Merged
Dec 26, 2024 | Dec 29, 2024

Link: https://github.com/networkx/networkx/pull/7779

This was my fourth contribution to the NetworkX library. While exploring subgraph monomorphisms, I noticed that the example provided did not accurately reflect the function's behavior. After reviewing its actual functionality, I updated the documentation to correct the wording and improve clarity.

This PR was successfully merged.

#5 Contribution

Started | Merged
Mar 6, 2025 | Mar 25, 2025

Link: https://github.com/networkx/networkx/pull/7901

This contribution was based around a medium to hard issue. The goal of this contribution is to resolve a problem in `chordless_cycles()`, where multigraphs with self-loops were not being handled correctly.

First, I revisited the NetworkX documentation and thoroughly analyzed the code line by line using multiple examples to understand its nooks and corners.

This contribution required me to dive into edge cases. Specifically, I designed scenarios using multigraphs with self-loops and parallel edges to observe how the function behaved. The real challenge emerged when I added a tricky test case involving the handling of two-cycles in a multigraph. While the initial tests only highlighted issues with self-loops, this new test case exposed a deeper flaw in the function's logic.

Dan Schult (dschult) guided me through this.
This contribution made the following changes to the project:

- Added tests that checked :
    - ❖ Self-loops as single-node chordless cycles.
    - ❖ Parallel edges forming valid two-node cycles.
    - ❖ Proper exclusion of self-loop nodes from larger cycles.

- A new parameter was introduced to track self-loop nodes and ensure that the graph `F` excluded them. The previous implementation correctly avoided adding node `u` to `F` if it had a self-loop. However, it did not check whether `u` was ignored due to a self-loop before adding its adjacent node `v` to `F`. This fix applied the same validation to `v`, preventing self-loop nodes from incorrectly contributing to larger cycles.

This change was successfully merged.

#6 Contribution

Started | Merged
Mar 18, 2025 | Mar 20, 2025

Link: https://github.com/networkx/networkx/pull/7910

This was my sixth contribution to the library. The function `minimum_st_node_cut()` incorrectly returned an empty dictionary `{}` instead of an empty `set()` when the source and target nodes (`s` and `t`) were directly connected by an edge or when the source and the target were in entirely different components.

I explored how `minimum_st_node_cut()` works by reviewing the function's code and its purpose in determining node cuts.
This contribution made the following changes to the project:

- Added a docstring explaining that the function returns an empty set when the source and target nodes are either in different components or are directly connected by an edge, as no node removal can destroy the path.

- Corrected the return type in the function by replacing `{}` with `set()` to ensure consistency with the documented return type.

- Adjusted the assertion statement in the relevant test cases from assert `nodelist == {}` to assert `nodelist == set()` to reflect the correct expected output.

This change was successfully merged.

Contributions to Nx-parallel

#7 Contribution

Started | Merged
Mar 24, 2025 | Mar 26, 2025

Link: https://github.com/networkx/nx-parallel/pull/100

This was my seventh contribution to the library and my first contribution to nx-parallel. When the methods (`betweenness_centrality()` & `edge_betweenness_centrality()`) were called on an empty graph, it resulted in a `list index out of range` error due to the absence of nodes or edges.

This contribution made the following changes to the project:

- Added a check to verify if the input graph is empty, returning an empty dictionary `{}` instead of proceeding with further computations.

This change has been successfully merged. I am currently working on implementing this check in other methods of nx-parallel if they lead to a similar error.

#8 Contribution

Started | Open
Mar 30, 2025 | N/A

Issue raised: https://github.com/networkx/nx-parallel/issues/94

While exploring the **nx-parallel** codebase, I encountered an issue where `get_all_functions()` in `test_get_chunks.py` was incorrectly displaying function details as `{'args': [], 'kwargs': 'args'}` instead of the expected function signatures. After investigating the issue, I raised an issue to check with the mentors if this behavior is consistent with the intended implementation or if there is something I might have overlooked in my understanding.

Based on suggestions, I investigated the role of decorators and their potential impact on Python's `inspect` methods. I tested this by commenting out decorators, but the issue persisted, leading me to suspect other underlying causes in the code. Further research into Python's inspection functions led me to a key finding— Use inspect.signature() instead of inspect.getfullargspec().

Pull Request Link: https://github.com/networkx/nx-parallel/pull/104

This was my seventh contribution to the library, and the following changes were made to the project:

- Replacing the use of `inspect.getfullargspec()` with `inspect.signature()` for extracting function arguments.
- Modified the assertion loops to iterate over edges instead of nodes for methods in `chk_dict_vals`, using the `.get(edge, 0)` method to properly access edge values from the dictionaries being compared.

After implementing the recent changes, I verified that the `get_all_functions()` method now works as expected.
The pull request addressing this change is currently pending review.

Past Projects

1.Previously-On-Netflix Web Application

Github Repo: Previously-On-Netflix

Previously on Netflix is a web-based application developed to offer users a personalized and insightful look into their Netflix viewing habits. Built using Python (Flask), HTML, CSS, and JavaScript, this project was inspired by the popularity of features like Spotify Wrapped, which are highly anticipated year-end experiences. Recognizing that everybody is especially drawn to such personalized recaps, this application brings a similar experience to Netflix users.

Users can upload their Netflix viewing history, which the system then parses and analyzes to generate visual summaries of their watch time, top shows or movies, favorite genres, and viewing patterns across different timeframes. I integrated data from IMDb and TMDB to enrich the experience with top-rated titles, genre metadata, and content details. I also implemented a content-based filtering system to provide personalized recommendations — saving time otherwise spent browsing. By integrating metadata from TMDB and IMDb, each show or movie was represented as a feature vector based on attributes like genre, cast, and description. Then I computed the cosine similarity between the items in a user's history and other titles in the dataset, allowing the app to recommend shows that are most similar to what the user has previously enjoyed.

It includes 2 other features namely The Roast Bot and Netflix Blend. The Roast Bot humorously critiques users' viewing habits with light-hearted commentary, making the insights more engaging and fun and the Netflix Blend feature allows users to compare their viewing history with friends', highlighting their shared interests. I calculated the Jaccard similarity between their sets of watched titles and genres to generate a taste match score and displayed shared shows or genres as common interests.

The backend is powered by Flask for data processing and routing, while JavaScript has been used for dynamic content updates and interactive visuals on the frontend. Through this project, I gained practical experience in full-stack development, API integration, and building data-driven, user-centric web applications.

2.Signify-Me

GitHub Repo: Signify-Me

SignifyMe is a web-based application designed to translate American Sign Language (ASL) into text using AI-driven hand gesture recognition. The goal of the project was to create an accessible tool that can interpret ASL in real time or from uploaded videos, making communication more inclusive and bridging the gap between the hearing and non-hearing communities. This was a collaborative project developed by a team of three, where we worked closely on both the AI model and the web interface.

The development process began with creating a machine learning model using the MediaPipe library. The first step involved feature extraction, where I used MediaPipe to detect 21 key hand landmarks (each with x, y, z coordinates and visibility scores). These coordinates served as the raw features that represent hand positions in space, effectively encoding the gesture.

Once the dataset was generated with labeled data (where each row represented a known ASL gesture), I used supervised learning to train the model. It allowed the model to learn mappings from hand landmark patterns to gesture labels. After data-preprocessing, I trained the model using classification algorithms : Logistic Regression, Random Forest and K-Nearest Neighbors (KNN).

I integrated the model with a Python script that, using MediaPipe and OpenCV, could capture real-time camera input and predict gestures with associated probabilities.

Through building this project, I gained valuable experience in developing a machine learning model and full-stack web development.

Project Details

Background and Research

Nx-parallel is a backend for NetworkX designed to implement and optimize parallel graph algorithms using Joblib. While NetworkX does a pretty good job handling complex graph operations, its performance can become a bottleneck when dealing with big graphs or computationally expensive algorithms. To address this, nx-parallel provides parallel implementations of NetworkX algorithms, leveraging multi-core processors for significant performance improvements. By dividing tasks into smaller, independent units, embarrassingly parallel algorithms can be executed simultaneously across multiple cores, which reduces execution time and enhances scalability.

Pre-proposal work

I have begun my work on the `triangles()` function in `cluster.py`.

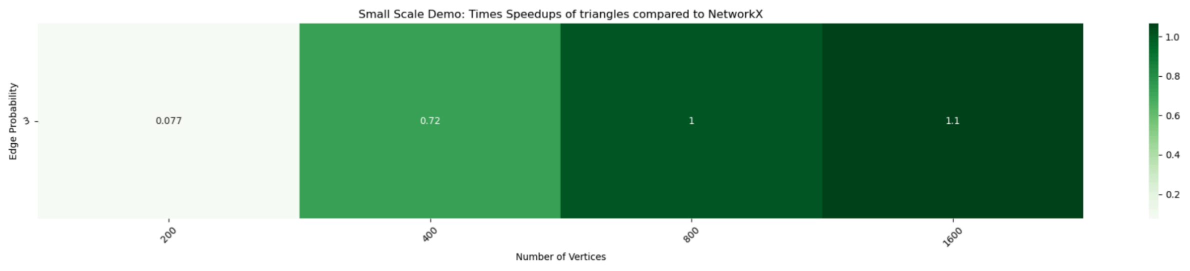
The time complexity of the triangle counting algorithm is $O(n \times d^2)$, where n is the number of nodes and d is the average degree of nodes. This results in long processing times when a large number of nodes are involved. By distributing across multiple cores in a processor with multiple cores, execution time can be reduced. Since the triangle formation around each node is independent of other nodes, the problem is embarrassingly parallel.

I started by thoroughly reviewing the current implementation of `triangles()` in `cluster.py`. I separated the parts of the code responsible for counting triangles and located the most computationally expensive operations.

Pull Request: https://github.com/networkx/nx-parallel/pull/106

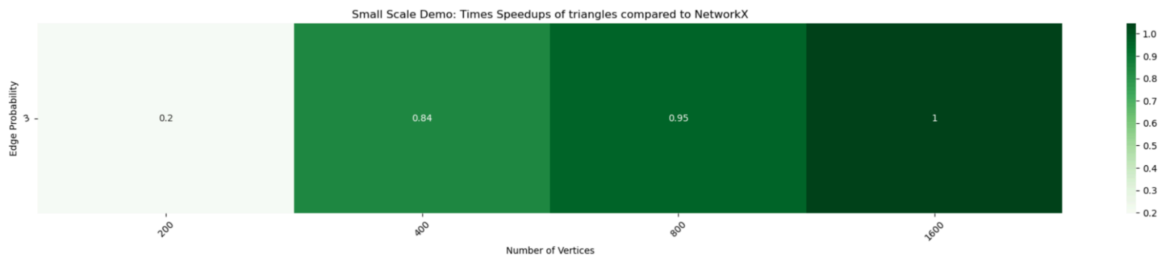
The main functionality of the triangle-counting algorithm has been moved into the inner function `compute_triangles_chunk()`, which processes a chunk of nodes at a time. Instead of handling each node independently across all CPU cores, I first divided the set of nodes into chunks based on the `n_jobs` (number of available jobs) parameter. Each chunk is then processed in parallel, where `compute_triangles_chunk()` iterates over its assigned nodes and counts the respective number of triangles.

The results from each parallel computation are stored in a Counter and, after all jobs complete, the partial results are merged into a final Counter to produce the complete triangle count per node. To better understand the efficiency of the parallelized triangle-counting algorithm, I generated a heatmap that illustrates the execution time across different numbers of processes.



From the heatmap, I observed:

1. For small graphs, the parallel implementation shows a speedup < 1, which seems to align with my understanding because of overhead of parallel execution and unnecessary task distribution.
2. For medium and large graphs, it starts outperforming the sequential approach.
3. Increasing the number of parallel jobs beyond a certain point does not always lead to better performance. The heatmap shown earlier was generated with the number of jobs manually set to 4. However, when I increased it to 6, the results did not improve as expected :



Update the Heatmap Benchmarking Script:

The existing script for generating heat maps is outdated and produces inconsistent timing results, as noted in Issue #51. As part of this project, I plan to revise and standardize the heatmap benchmarking script to ensure it accurately reflects the execution time of each function.

I have begun by thoroughly examining the current script to understand how it records execution times, what metrics it logs, and how these are used to generate heatmaps. From this analysis, it is clear that the script currently relies on `time.time()`, which is susceptible to fluctuations caused by background processes and system load.

I have my end semester examinations from 20th April - 2nd May, so I will not be available for these 13 days. After that my break begins and I will explore Python's timeit module.

The approach I followed for implementing `triangles()` will also be applied to the below algorithms :

Harmonic Centrality

The Harmonic Centrality of a node u measures the sum of the reciprocal of the shortest path distances from all other nodes to u. This is an embarrassingly parallel task because the computation of shortest paths from each of the source nodes is entirely independent of the others. The parallelization strategy involves the distribution of chunks of nodes across multiple cores, each processing a subset.

Clustering

Clustering Coefficient computes the tendency of a node's neighbors to form triangles. It iterates over all nodes, computing triangle counts and degree values. Each of these computations for each node are fully independent of one another, as the triangle count around a node relies solely on its local neighbourhood making it embarrassingly parallel.

Jaccard Coefficient

The Jaccard Coefficient measures node similarity as the ratio of common to total neighbors. This coefficient is computed for given node pairs independently, making it embarrassingly parallel due to the lack of inter-pair dependencies.

Average Neighbor Degree

Average Neighbor Degree computes the mean degree of neighbors for each node. Since each node's average is computed independently based on its neighborhood, the function is highly parallelizable. The nodes will be divided across workers, and their local averages will be computed in parallel. The results will be merged efficiently, with attention to edge weights and directionality where applicable.

Project Timeline

May 8th - June 1st (Community Bonding Period)	• Review relevant documentation on joblib and parallelization concepts. • Work on the heatmap benching script. • Connect with mentors to understand expectations and clarify doubts. • Select the remaining algorithms to parallelize.
Week 1 **June 2th - June 8th**	• Implement the first algorithm. • Conduct initial tests and performance benchmarks. • Get feedback from mentors.
Week 2 **June 9th - June 15th**	• Add the suggestions recommended. • Finish working on the first algorithm. • Research and implement the second algorithm.

Week 3 **June 16th - June 22th**	● Conduct initial tests and performance benchmarks. ● Get feedback from mentors.
Week 4 **June 23rd - June 29th**	● Add the respective recommendations. ● Finish working on the second algorithm.
Week 5 **June 30th - July 6th**	● Research and implement the third algorithm. ● Conduct initial tests and performance benchmarks. ● Get feedback and continue working on the third algorithm.
Week 6 **July 7th - July 13th**	● Finish working on the third algorithm. ● Buffer week for improvements and debugging.
Week 7 **Mid-Term Evaluation** **July 14th - July 20th**	● Submit progress for mid-term evaluation. ● Research and implement the fourth algorithm.
Week 8 **July 21st - July 27th**	● Conduct initial tests and performance benchmarks. ● Get feedback and finish working on the fourth algorithm.
Week 9 **July 28th - August 3rd**	● Research and implement the fifth algorithm. ● Conduct initial tests and performance benchmarks.

Week 10 **August 4th - August 10th**	● Get feedback and finish working on the fifth algorithm. ● Research and implement the sixth algorithm. ● Conduct initial tests and performance benchmarks.
Week 11 **August 11th - August 17th**	● Get feedback and finish working on the sixth algorithm. ● Buffer week for improvements and suggestions.
Week 12 **August 18th - August 24th**	● Research and implement the final algorithm. ● Conduct initial tests and performance benchmarks. ● Get a review and finish working on the seventh algorithm.
Final Evaluation **August 25th - August 31st**	● Conduct final tests to verify the correctness and efficiency of all the algorithms inculcating any final suggestions and recommendations. ● Create a final project report and submit it with all the work done for a final review.

Throughout the internship timeline, the work will be continuously shared with the mentors for feedback and reviews in the form of blogs. Longer monthly reports or half-monthly reports, whichever is preferable, would be presented for tracking the progress of the project comprehensively. I will push my daily progress in the code to GitHub to allow the mentors to review it easily and provide feedback and comments for improvements. These check-ins would assist us to stay on track and document the progress made in the course of the internship.

I would also stay active on the communication channels like Discord (or any other channel which is agreed upon in the course of the internship) to enable day-to-day one-on-one conversations with the mentors. I am also available for weekly or monthly video chats to share updates about my work and resolve issues in the project.

Time Availability

The Google Summer of timeline is mostly in sync with my university's summer break and thus will allow me ample time to work on my project. Even after the break ends, there would be no tests or examinations, and the classes would be asynchronous, leaving me enough time to work. My awake hours are between 11:00 AM IST (5:30 AM UTC) and 12:00 AM IST (6:30 PM UTC), and I would be reachable any time between them. If I take a day off due to unexpected situations, I will inform the mentors in advance and will make up for that time in the subsequent days of the internship.

I will have my university classes from August and their timings are 11:00 AM IST (5:30 AM UTC) to 04:00 PM IST (10:30 AM UTC). During this month, I will be available to work on the project at any time except the time I have my classes. But I would still be reachable during that time.

Post GSoC

Contributing to NetworkX over the past few months has helped me discover how much I enjoy working with algorithms and trying to find the most efficient solutions. I will continue to contribute to NetworkX even after GSoC’25. I’m looking forward to growing through this experience.

Other Commitments

NetworkX is the only organization I’m applying to for GSoC’25 because it truly aligns with my interests and goals. I’ve enjoyed contributing so far, and I’m excited about the potential to work more closely on something I care deeply about. By focusing entirely on this project, I’ll be able to give it my full attention.