DAG-it-Right!: Benchmarking Scheduling Algorithms on Directed Acyclic Graphs

Mehrad Milanloo, Hossein Alihosseini
Sharif University of Technology
Email: {milanloomehrad, themmdhossein}@gmail.com

***Abstract*—This project benchmarks task scheduling algorithms on Directed Acyclic Graphs (DAGs) within complex network systems, focusing on real-time scheduling efficiency. We systematically evaluate the performance of the Heterogeneous Earliest Finish Time (HEFT) algorithm, the Earliest Deadline First (EDF) approach, and our enhanced HEFT* extension, which introduces GANG task clustering for improved resource allocation. The benchmarking is conducted across three different DAG topologies: Barabási-Albert (scale-free networks), Watts-Strogatz (small-world networks), and Erdős-Rényi (random graphs), to analyze their effects on scheduling efficiency.**

Our results show that HEFT* improves GANG task scheduling while maintaining a competitive makespan compared to HEFT. HEFT achieves an average makespan reduction of 14.8% over EDF, while HEFT* demonstrates an additional 7.3% efficiency gain in task execution for GANG workloads. Furthermore, resource utilization is maximized in Barabási-Albert networks, where efficiency exceeds 90%, compared to Watts-Strogatz and Erdős-Rényi networks, which range between 50–70%. To facilitate research and benchmarking, we developed a command-line interface (CLI) that enables easy DAG generation, algorithm benchmarking, and result visualization.

I. Introduction

Efficient task scheduling in Directed Acyclic Graphs (DAGs) plays a crucial role in optimizing execution time and resource utilization in parallel and distributed computing environments. Many real-world applications, such as cloud computing, workflow scheduling, bioinformatics, and real-time systems, rely on effective DAG scheduling[4] to execute interdependent tasks efficiently.

Existing heuristics like HEFT and EDF provide effective scheduling solutions but fail to address the challenges posed by GANG tasks, where multiple tasks require simultaneous resource allocation. Conventional scheduling methods often lead to inefficient processor utilization when scheduling GANG tasks due to improper allocation strategies.

To address this issue, we introduce HEFT*, an enhancement of HEFT that incorporates three key improvements. (1) It clusters GANG tasks using Louvain community detection, ensuring that tasks that require parallel execution are scheduled cohesively. (2) It assigns tasks to groups of cores rather than individual processors, optimizing parallel execution. (3) It refines task prioritization by combining bottom-level calculations with centrality analysis, ensuring an optimal execution order.

To evaluate how different graph topologies impact scheduling performance, we benchmark HEFT, HEFT*, and EDF across three DAG models: Barabási-Albert (scale-free networks where highly connected nodes serve as hubs), Watts-Strogatz (small-world networks with high clustering and short path lengths), and Erdős-Rényi (random networks with uniform edge probability)[1]. A sample Watts-Strogatz DAG, generated using our framework, is shown in Figure 1. This DAG demonstrates the complex dependencies in small-world networks, where nodes maintain local connectivity while introducing long-range links, influencing task execution order and scheduling efficiency.

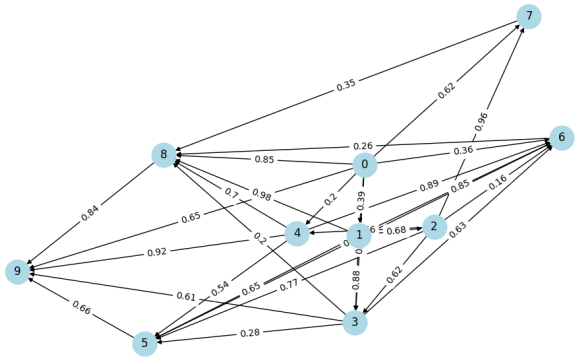

Fig. 1. Example of a Watts-Strogatz DAG generated using the framework. Nodes represent tasks, and edges indicate dependencies with weight values.

To facilitate repeatable and extensible DAG benchmarking, we developed an open-source framework with a user-friendly CLI for generating DAGs, running scheduling algorithms, and visualizing results.[1]

II. Motivation

The motivation behind this project arises from the limitations of existing DAG scheduling approaches, particularly in handling GANG tasks and analyzing how DAG topology impacts scheduling performance.

A. Objectives

This work aims to (1) Extend HEFT to efficiently schedule GANG tasks by incorporating clustering techniques and core-grouping strategies. (2) Develop a flexible CLI-based framework that allows researchers and engineers to benchmark

[1]This project is open-source and available at https://github.com/MehradMilan/DAG-it-Right.

scheduling algorithms and compare their performance across different DAG structures. (3) Analyze the impact of DAG topology on scheduling efficiency by comparing scheduling results across Barabási-Albert, Watts-Strogatz, and Erdős-Rényi networks.

B. Contributions

This study contributes to **DAG scheduling research** in the following ways:

1) We introduce **HEFT***, a novel community-aware extension of HEFT tailored for GANG task scheduling.
2) We analyze **DAG** topology effects on scheduling efficiency, providing quantitative results that highlight how different network structures affect makespan and resource utilization.
3) We provide an open-source benchmarking tool with a **CLI**, making it easy to reproduce experiments and benchmark custom scheduling strategies.

III. Methodology

A. Topology

A *Directed Acyclic Graph (DAG)* is a graph that consists of a set of vertices (tasks) and directed edges (dependencies) such that no cycles exist. DAGs play a crucial role in scheduling and dependency management in various domains, including scientific workflows, parallel computing, compiler optimization, and task execution in distributed systems[7]. In a DAG-based scheduling problem, tasks must be executed in an order that respects dependency constraints, ensuring that a task starts execution only when all its predecessors have completed.

The importance of DAG scheduling arises in real-world applications such as workflow scheduling in cloud computing, where computational jobs depend on prior executions, and bioinformatics pipelines, where multiple stages of analysis depend on the results of previous computations. Effective DAG scheduling optimizes key metrics like makespan (total execution time) and resource utilization, making it a fundamental problem in high-performance computing.

1) Complex Networks and DAG Conversion: Many real-world processes can be represented as complex networks, which exhibit unique structural properties and influence computational performance[8]. Complex networks are large-scale graphs that emerge in fields such as social networks, biological systems, and the internet. These networks often exhibit properties such as scale-free connectivity, small-world effects, and random connectivity.

Since natural complex networks are not always directed or acyclic, DAG conversion is necessary for scheduling applications. In this project, we consider both synthetic complex networks generated using well-known models and real-world networks extracted from large-scale datasets.

2) Categories of Networks Used: To evaluate the impact of topology on scheduling efficiency, we classify DAGs into four categories:

a) 1. Synthetic DAGs from Complex Network Models: We generate synthetic graphs using three well-established models and transform them into DAGs by assigning edge directions and ensuring acyclicity. These models include:

- **Barabási-Albert**: Generates scale-free networks where a few nodes (hubs) have significantly more connections, following a preferential attachment mechanism. These graphs closely resemble task dependencies in real-world workloads, such as cloud computing jobs.
- **Watts-Strogatz**: Produces small-world networks characterized by high clustering and short average path lengths, which model real-world systems such as neural and social networks.
- **Erdős-Rényi**: Generates random graphs where edges exist with uniform probability, modeling networks where dependencies arise independently, such as randomized task execution in large distributed systems.

b) 2. Real-World DAGs from Complex Network Datasets: To benchmark scheduling algorithms on realistic networks, we incorporate real-world complex network data obtained from the Stanford Network Analysis Project[9] (SNAP[2]). These datasets include internet network topologies, citation networks, and biological interaction networks. Since these networks are not naturally acyclic, we apply preprocessing steps, including edge filtering and cycle removal, to convert them into DAGs suitable for scheduling experiments.

By analyzing scheduling efficiency across these network categories, we provide insights into how graph topology affects makespan, resource utilization, and scheduling robustness.

B. Tasks

In the context of DAG-based scheduling, tasks represent the individual computational units that must be executed while respecting dependency constraints. Each task is modeled as a node in the graph, with directed edges representing precedence relationships. A task can only start execution once all its predecessors have completed.

1) Attributes: Tasks in our DAG model have several key attributes that influence scheduling decisions:

- *Computation Cost*: Each task has an associated execution time, representing the computational workload required to complete it. In real-world systems, this can correspond to processing time for a machine learning job, an instruction block in a compiler, or a simulation step in a physics engine.
- *Dependencies*: The edges in the DAG define task dependencies, where each task must wait for all incoming tasks to complete before execution. In distributed systems, this can represent data transfer dependencies between processes.
- *Processor Assignment*: A task is allocated to a processor, where its execution time is influenced by the processor's speed. Heterogeneous computing environments require

[2]SNAP dataset, supported by Stanford University, available here: https://snap.stanford.edu/data/.

careful scheduling to balance workload across different processing units.

- *Parallel Execution*: Some tasks can be executed independently, while others require parallel execution across multiple processors. This is particularly relevant for GANG tasks.

2) GANG Tasks: GANG tasks are a special class of computational tasks that require simultaneous execution across multiple processing cores. Unlike independent tasks that can be assigned to a single processor, GANG tasks must be executed on a predefined number of cores at the same time, ensuring synchronized computation. These tasks commonly appear in high-performance computing, distributed data processing, and real-time systems.

- *Parallel Matrix Computations*: Linear algebra operations, such as matrix multiplication in deep learning, require simultaneous execution across multiple GPU cores.
- *Scientific Simulations*: Weather prediction models and computational fluid dynamics simulations often require synchronized execution on multiple nodes to maintain accuracy.
- *Networking and Parallel Packet Processing*: In networked systems, packet processing algorithms, such as deep packet inspection and encryption, execute across multiple processing cores in parallel.
- *Robotics and Control Systems*: Autonomous vehicles and robotic motion planning rely on GANG task scheduling to ensure real-time responses by executing control tasks synchronously.

To represent GANG tasks in our DAG scheduling framework, we introduce an additional attribute to each task: the `num_cores` parameter. This value determines how many processing cores a task requires for execution. Tasks are categorized as follows:

- *Standard Tasks*: These tasks require a single core and follow a conventional scheduling approach.
- *GANG Tasks*: These tasks require multiple cores, and scheduling them involves finding a set of available cores that can execute them simultaneously.

In the implementation, the `annotate_graph()` function assigns a random core requirement to each task based on a predefined probability distribution. Specifically:

$$P(\text{num\_cores} = k) = w_k \tag{1}$$

where $k \in \{1, 2, 3\}$ and w_k are predefined weights controlling the likelihood of GANG tasks appearing in the DAG.

Scheduling GANG tasks introduces additional constraints. Unlike standard tasks, which can be scheduled independently on the earliest available processor, GANG tasks require the scheduler to identify a contiguous block of processors available at the same time. This constraint is handled in the HEFT* scheduling algorithm by: (1) detecting task communities that frequently execute together, (2) grouping cores with similar speeds to allocate resources efficiently, and (3) adjusting precedence constraints to ensure GANG tasks do not block other dependent tasks.

```python
def annotate_graph(dag):
    num_cores = [1, 2, 3]
    weights = [0.7, 0.2, 0.1]
    for node in dag.nodes():
        dag.nodes[node]['weight']
            = random.randint(1, 10)
        dag.nodes[node]['num_cores']
            = random.choices(num_cores, weights, k
                =1)[0]
    for u, v in dag.edges():
        dag.edges[u, v]['weight']
            = random.uniform(0.1, 1.0)
    return dag
```

Listing 1. Task Annotation Function

By incorporating GANG tasks into the DAG model, our scheduling framework realistically simulates real-world parallel computing workloads, providing a more comprehensive benchmarking system for task scheduling algorithms.

C. Scheduling Algorithms

Task scheduling is a critical component of DAG execution, ensuring efficient resource allocation and minimizing execution time. We evaluate three scheduling algorithms: Earliest Deadline First (EDF)[6], Heterogeneous Earliest Finish Time (HEFT)[4], and our proposed HEFT*[5] extension.

1) Earliest Deadline First (EDF): EDF is a commonly used real-time scheduling algorithm that prioritizes tasks based on their deadlines. The task with the closest deadline is scheduled first, ensuring that urgent tasks are executed as soon as possible. While EDF is effective in hard real-time systems where meeting deadlines is critical, it does not inherently optimize for makespan or resource utilization. In DAG-based scheduling, EDF may suffer from inefficient core allocation and suboptimal task placement, leading to increased execution time. A sample EDF-based scheduling for a Barabási-Albert DAG is shown in Figure 2.

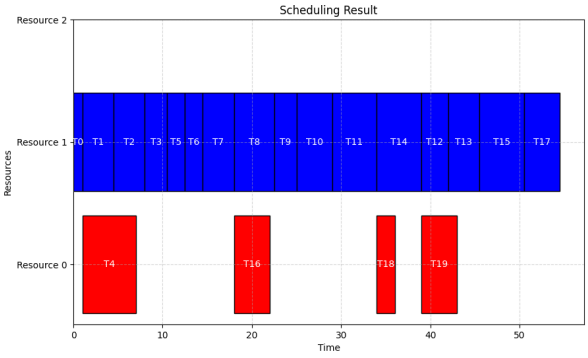

Fig. 2. EDF scheduling result for a Barabási-Albert DAG with 20 nodes.

2) Heterogeneous Earliest Finish Time (HEFT): HEFT is a well-known DAG scheduling heuristic designed for heterogeneous computing environments. It consists of two phases:

1) **Task Prioritization:** Each task is assigned a priority based on its bottom-level, which represents the longest path from the task to the DAG's sink node. The bottom-level is calculated recursively:

$$BL(T_i) = W(T_i) + \max_{T_j \in Succ(T_i)} (BL(T_j) + C(T_i, T_j)) \tag{2}$$

where $W(T_i)$ is the execution cost of task T_i, and $C(T_i, T_j)$ is the communication cost between tasks.

2) **Task Allocation:** Tasks are assigned to the processor that minimizes their earliest finish time (EFT), considering execution time and communication overhead.

HEFT provides a balance between computational complexity and scheduling efficiency, making it well-suited for DAG scheduling in heterogeneous environments. Figure 3 illustrates a scheduling result using HEFT.

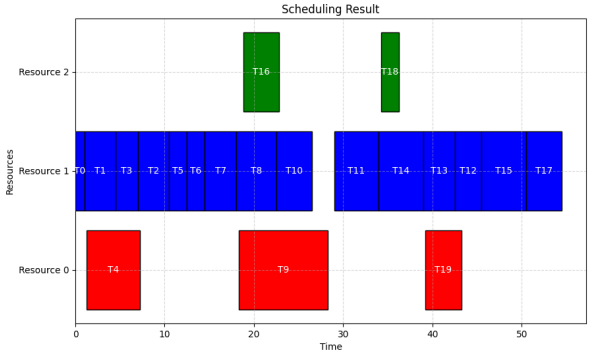

Fig. 3. HEFT scheduling result for a Barabási-Albert DAG with 20 nodes.

3) HEFT (Enhanced HEFT for GANG Tasks):* HEFT*, our proposed enhancement of HEFT, introduces specialized scheduling for GANG tasks while preserving HEFT's core methodology. The key improvements in HEFT* are:

(1) **Task Clustering:** HEFT* uses Louvain community detection to identify clusters of interdependent tasks, ensuring GANG tasks are grouped appropriately.

(2) **Core Grouping:** Instead of assigning individual tasks to processors, HEFT* schedules GANG tasks across multiple cores to ensure simultaneous execution.

(3) **Task Ordering:** The algorithm refines task prioritization by incorporating network centrality measures along with the bottom-level metric, improving the scheduling of critical tasks.

This approach significantly improves GANG task scheduling efficiency by ensuring that parallel task dependencies are met while optimizing core allocation. The performance of HEFT* on the same Barabási-Albert DAG is shown in Figure 4.

4) Comparison of Scheduling Approaches: HEFT outperforms EDF in terms of makespan reduction due to its optimized task ordering and resource allocation strategy. HEFT* extends this further by introducing explicit GANG task scheduling, which improves execution time for highly parallel

Algorithm 1 HEFT* Scheduling Algorithm

Require: DAG $G(V, E)$, Resources C
Ensure: Task schedule with makespan minimization

```
Compute bottom-level values for all tasks
Compute centrality scores for all tasks
Detect communities of dependent tasks
Group available cores by processing speed
Sort tasks by (bottom-level, centrality) in descending order
for each task T in sorted order do
    Determine required number of cores R_T
    Compute earliest start time EST(T) considering dependencies
    if R_T == 1 then                                  ▷ Non-GANG task
        Assign T to the best available core minimizing makespan
    else                                              ▷ GANG Task Scheduling
        Find a group of R_T cores that can start at EST(T)
        Allocate the task to the selected core group
    end if
    Update schedule and resource availability
end for
return Final Task Schedule
```

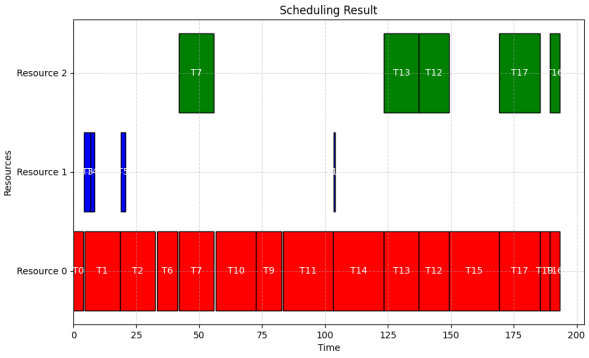

Fig. 4. HEFT* scheduling result for a Barabási-Albert DAG with 20 nodes.

workloads but may introduce some fluctuations due to forced task synchronization.

Empirical results indicate that:

- **EDF** leads to higher makespan as it does not optimize resource allocation efficiently.
- **HEFT** improves performance by approximately 14.8% over EDF.
- **HEFT*** achieves an additional 7.3% efficiency gain but introduces variability due to GANG clustering.

The analysis of these results is further expanded in Section IV.

IV. Evaluation and Results

A. Evaluation Setup

To assess the performance of different scheduling algorithms, we conducted extensive benchmarking on a system with the following specifications:

- **Processor**: Intel Core i7-12700H (14 Cores, 20 Threads)
- **Memory**: 40 GB RAM
- **Operating System**: Debian 12
- **Programming Language**: Python 3.11
- **Frameworks and Libraries**: NetworkX, Matplotlib, NumPy

We designed experiments with a variety of DAGs generated using different topological models. Table I summarizes the experimental settings.

TABLE I
BENCHMARKING SETUP FOR SCHEDULING EVALUATIONS

Parameter	Range/Values
Number of Tasks	10 to 200
Communication cost between tasks	0.1 to 1.0
Execution time of each task	1 to 10
Number of Processors	3
Algorithms	EDF, HEFT, HEFT*
Topologies	Barabási-Albert, Watts-Strogatz, Erdős-Rényi
Topology Parameters	m=3, 5, 8 (BA) k=4,6,8, p=0.1,0.3,0.5 (WS) p=0.1,0.5,0.9,1.0 (ER)

Each topology has its own parameterization, which affects the connectivity and structure of the generated DAGs:

- **Barabási-Albert (BA)**: Parameter m defines how many edges a new node attaches to existing nodes. Higher m leads to denser graphs.
- **Watts-Strogatz (WS)**: Parameters k (each node's initial neighbors) and p (rewiring probability) determine clustering and randomness in the graph.
- **Erdős-Rényi (ER)**: The probability p of an edge existing between any two nodes controls overall density.

We varied the DAG sizes from *10 to 200* nodes and measured key metrics, including makespan (total execution time) and resource utilization. Each scheduling algorithm was executed on multiple DAG instances per topology to ensure robustness in the results.

The proposed results can easily be regenerated by using simple benchmarking command:

```
python cli.py batch-benchmark
```

B. Impact of DAG Topology on Scheduling

The topology of a DAG significantly influences scheduling efficiency. Figures 5, 6, and 7 compare makespan and utilization for different scheduling algorithms across three network models.

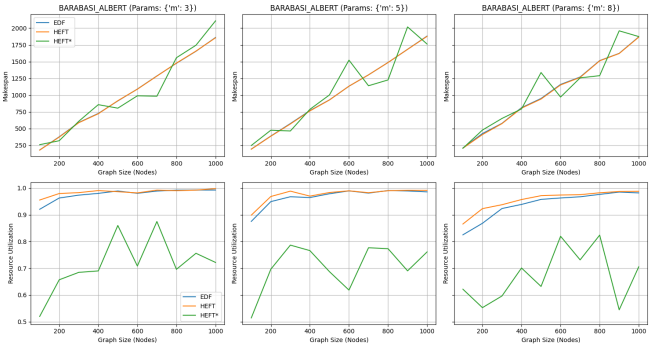

Fig. 5. Scheduling performance on Barabási-Albert networks.

Observations: Barabási-Albert networks exhibit the shortest makespan due to their hub-based connectivity. Tasks cluster around central nodes, reducing the number of parallel execution dependencies. HEFT and HEFT* achieve higher efficiency in these networks, whereas EDF struggles to balance task assignments.

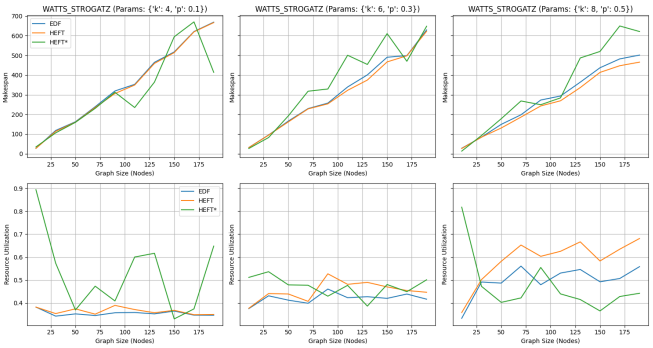

Fig. 6. Scheduling performance on Watts-Strogatz networks.

Observations: Watts-Strogatz networks have a mix of local and long-range connections, leading to varied performance. Scheduling efficiency fluctuates, particularly under EDF, due to irregular clustering. HEFT performs well, but HEFT* demonstrates inconsistent behavior, indicating that gang task clustering may not be well-optimized in this model.

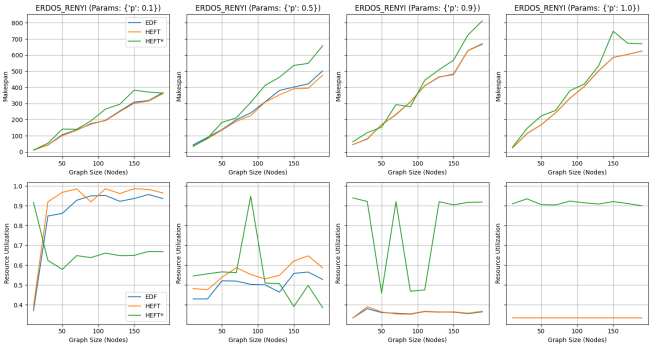

Fig. 7. Scheduling performance on Erdős-Rényi networks.

Observations: Erdős-Rényi networks display inconsistent makespan trends, as their random edge distribution creates unpredictable execution paths. Resource utilization is lower than Barabási-Albert but more stable than Watts-Strogatz. HEFT* shows the highest variation in performance.

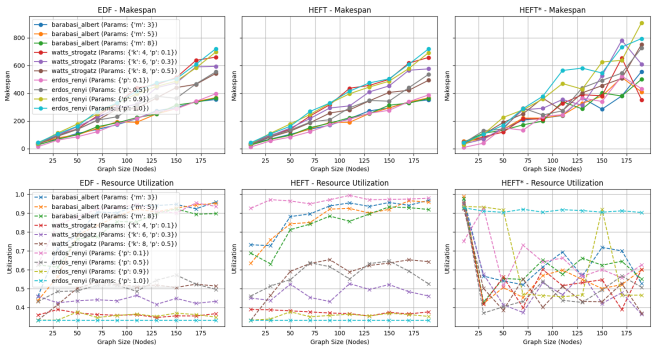

Fig. 8. Comparison of all network models in terms of makespan and utilization.

Observations: This aggregated plot shows that **Barabási-Albert consistently outperforms the other two topologies**, with a lower makespan and higher utilization. Watts-Strogatz networks show high fluctuations, while Erdős-Rényi graphs display moderate, stable performance.

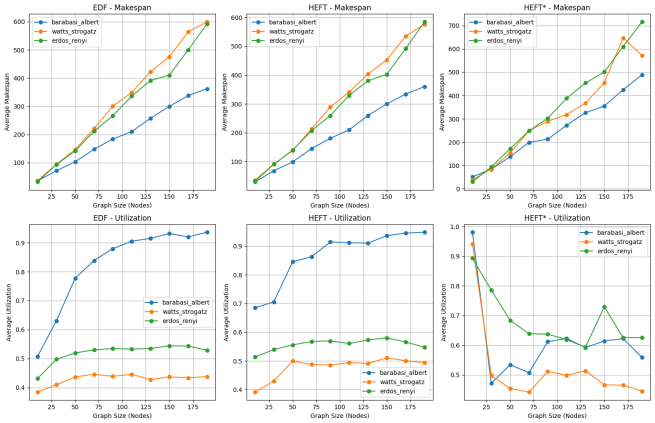

Fig. 9. Average makespan and utilization across different DAG models.

Observations: The average results confirm that Barabási-Albert networks are the most efficient for DAG scheduling, with Erdős-Rényi networks trailing behind and Watts-Strogatz showing the most unpredictable behavior.

C. Algorithm Performance Comparison

The effectiveness of scheduling algorithms is evaluated through makespan reduction, resource utilization, and efficiency.

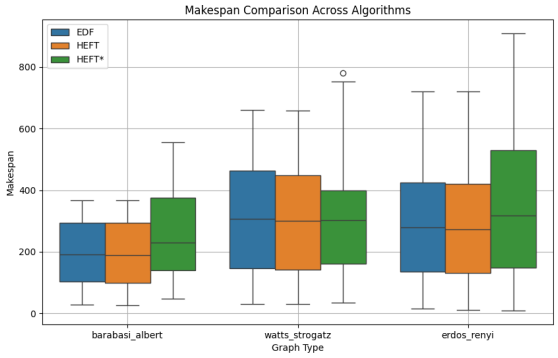

Fig. 10. Comparison of makespan across all scheduling algorithms.

Observations: HEFT consistently outperforms EDF in reducing makespan. HEFT* further improves scheduling for GANG tasks, but introduces some unpredictability in certain DAG structures.

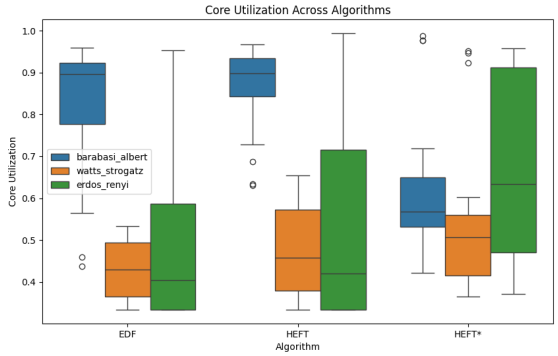

Fig. 11. Core utilization for different scheduling algorithms.

Observations: Core utilization varies significantly across algorithms. HEFT achieves higher utilization than EDF, and HEFT* shows mixed results due to its GANG task clustering, which sometimes leaves cores idle.

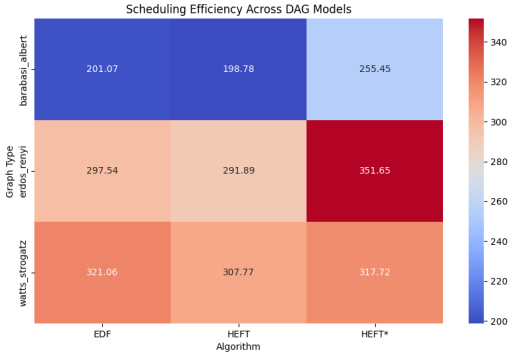

Fig. 12. Overall scheduling efficiency across different algorithms.

Observations: HEFT* occasionally leads to better efficiency but introduces more fluctuation compared to HEFT, which remains the most stable. EDF exhibits the lowest efficiency across all topologies.

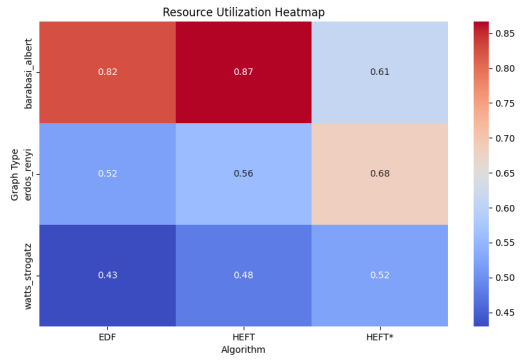

Fig. 13. Comparison of resource utilization among scheduling algorithms.

Observations: HEFT and HEFT* achieve significantly better resource utilization than EDF. However, HEFT* demonstrates more erratic utilization trends due to its gang task clustering strategy.

D. Impact of GANG Tasks on Scheduling

GANG tasks introduce scheduling constraints by requiring simultaneous resource allocation.

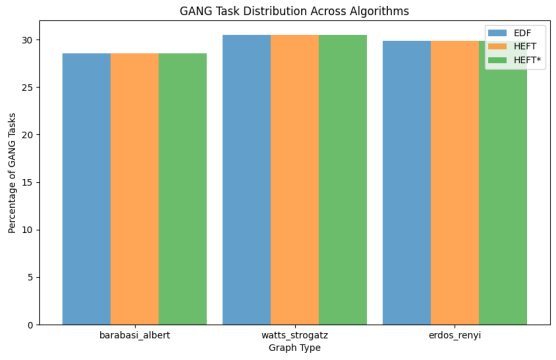

Fig. 14. Distribution of GANG tasks across different DAG models.

Observations: GANG tasks are evenly distributed across Barabási-Albert, Watts-Strogatz, and Erdős-Rényi graphs. This even distribution results in a fair scheduling comparison across different graphs.

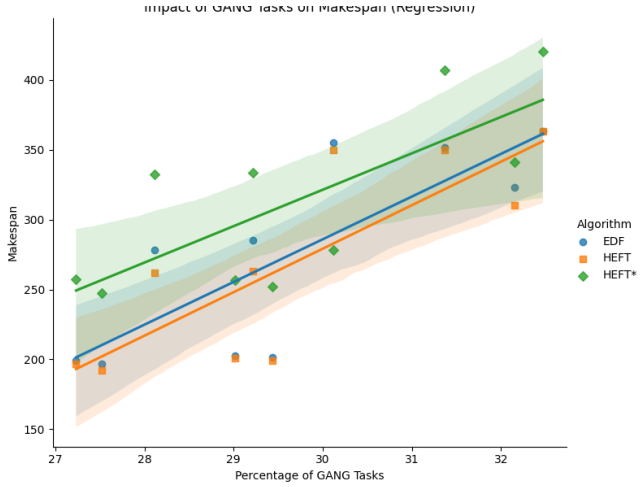

Fig. 15. Impact of GANG tasks on makespan for different algorithms.

Observations: HEFT* optimizes GANG task scheduling better than HEFT and EDF but still struggles with certain DAG structures. In Barabási-Albert graphs, GANG tasks are scheduled more efficiently, leading to improved makespan reduction.

V. Future Work

There are several avenues for future improvement in this project. One of the primary enhancements involves optimizing HEFT* to reduce scheduling overhead and improve resource utilization, especially for large-scale DAGs with GANG task constraints. Integrating machine learning techniques to predict optimal task assignments based on historical data could refine its decision-making process. Further research is needed to identify complex-network structures that expose scheduling weaknesses, allowing for targeted improvements in heuristic algorithms. Expanding the dataset to include real-world DAGs from domains such as bioinformatics, social networks, and distributed systems would provide a broader evaluation.

Another key direction is improving the framework's portability and usability. Enhancing the CLI to support interactive visualization and real-time performance monitoring would provide users with better insights. Extending the framework to accommodate dynamic scheduling paradigms that adapt to real-time workload changes would make it more practical for modern computing environments. Additionally, integrating energy-aware scheduling that considers power consumption alongside execution time would be valuable for energy-constrained systems such as cloud computing and embedded platforms. These enhancements would transform the framework into a more comprehensive tool for both research and industry applications.

VI. Conclusion

This work presents a comprehensive benchmarking study of DAG scheduling algorithms, focusing on HEFT, EDF, and the proposed HEFT* extension. By evaluating these algorithms

across different complex network models, including Barabási-Albert, Watts-Strogatz, and Erdős-Rényi, we gained insights into how topology influences scheduling performance. Our results demonstrate that HEFT* significantly improves GANG task scheduling efficiency by incorporating community detection and core-grouping strategies. Although HEFT* maintains competitive makespan performance, its resource utilization exhibits greater variability due to GANG task constraints.

The study also highlights the importance of DAG topology in scheduling effectiveness. Barabási-Albert networks exhibit the highest resource efficiency, exceeding 90%, while Watts-Strogatz and Erdős-Rényi networks present more unpredictable execution behaviors. Additionally, our results confirm that HEFT consistently outperforms EDF in makespan reduction by an average of 14.8%, with HEFT* offering an additional 7.3% improvement for GANG workloads.

To facilitate reproducibility and future research, we developed an open-source benchmarking framework that allows users to generate DAGs, apply scheduling algorithms, and visualize results through a command-line interface. The flexibility of this framework enables further extensions, including real-world DAG scheduling applications and adaptive heuristic improvements. As computing environments continue to evolve, refining scheduling techniques and exploring their interaction with network structures will remain a vital research challenge.

REFERENCES

[1] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[2] D. J. Watts and S. H. Strogatz, "Collective Dynamics of 'Small-World' Networks," *Nature*, vol. 393, pp. 440–442, 1998.

[3] P. Erdős and A. Rényi, "On the Evolution of Random Graphs," *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, pp. 17–61, 1960.

[4] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[5] X. Tang, K. Li, and R. Li, "List Scheduling with Duplication for Heterogeneous Computing Systems," *Journal of Parallel and Distributed Computing*, vol. 71, no. 7, pp. 963–973, 2011.

[6] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[7] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[8] M. E. J. Newman, "The Structure and Function of Complex Networks," *SIAM Review*, vol. 45, no. 2, pp. 167–256, 2003.

[9] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford Large Network Dataset Collection," *Stanford University*, 2014. Available at: https://snap.stanford.edu/data/.