EFFICIENT GRAPH COLORING ON LARGE GRAPHS

by

Mustafa Emir Uyar - 150120007
Muhammed Hayta - 150121068
Eren Duyuk - 150120509

CSE4197 Engineering Project 1

Project Specification Document

Supervised by:
Assoc. Prof. Betül Demiröz Boz

[illegible]

Marmara University, Faculty of Engineering

Computer Engineering Department

15.11.2024

1. PROBLEM STATEMENT

The graph coloring problem is a well-known NP-complete problem [1] in graph theory that presents significant computational challenges, particularly as the size of the graph increases. Solutions to this problem have crucial applications in diverse areas, including scheduling, network management, resource allocation, and other fields requiring efficient organization and distribution of resources [2-4]. Given its broad applicability, designing an optimized algorithm for graph coloring is a valuable and impactful contribution to computer science.

As graph sizes expand, graph coloring becomes increasingly difficult due to the exponential growth in potential coloring combinations and minimize conflicts within limited computational resources. Large-scale graphs often require innovative approaches that not only handle the exponential number of choices but also work within the constraints of memory, processing power, and runtime efficiency. Optimizing graph coloring for these extensive graphs is critical, as the computational challenges rise exponentially, yet current resources and methods for effectively coloring large-scale graphs remain limited.

2. PROBLEM DESCRIPTION and MOTIVATION

The graph coloring problem is an NP-complete problem in graph theory, where each node in a graph must be assigned a color such that no two adjacent nodes share the same color. As the size of the graph increases, solving this problem becomes increasingly complex and computationally demanding.

This challenge is significant for large-scale graphs with hundreds of thousands of nodes, which are commonly encountered in fields like network management, data scheduling, social network analysis, and resource allocation. Current literature lacks effective algorithms capable of solving the graph coloring problem on large graphs within a feasible time frame while achieving acceptable results.

In this project, we aim to fill this gap by developing an optimized approach to the graph coloring problem by using methods such as evolutionary algorithms, graph reduction techniques, bitwise implementation and parallelization. Through these methods, we hope to create a scalable and efficient solution for large-scale graph coloring, with the potential to enhance applications that rely on this fundamental problem.

3. MAIN GOAL and OBJECTIVES

The primary goal of this project is to develop a scalable and efficient evolutionary algorithm capable of solving the graph coloring problem for large-scale graphs through the application of bitwise operations, graph reduction, and parallelization techniques.

Objective 1:

Design an evolutionary algorithm that can consistently find solutions close to the best possible coloring for large graphs. The algorithm should effectively explore different coloring possibilities, creating a reliable tool that works well across various large graph types.

Objective 2:

Improve the algorithm’s efficiency by optimizing memory usage and reducing runtime, allowing it to handle very large graphs. This includes using bitwise operations, parallel processing, and efficient data structures to lower resource demands while keeping performance high.

Objective 3:

Conduct extensive benchmarking and testing on a diverse set of large graph datasets to thoroughly evaluate the algorithm's scalability and performance. Testing across various graph types and sizes will ensure that the algorithm performs at least comparably to existing algorithms in the literature, both in terms of execution efficiency and solution quality.

4. RELATED WORK

4.1 InCEA: Integrated Crossover-Based Evolutionary Algorithm for Coloring Vertex-Weighted Graphs [7]

The Integrated Crossover-Based Evolutionary Algorithm (InCEA) is designed to address the vertex-weighted graph coloring problem by efficiently minimizing the sum of weights for uncolored vertices within a fixed number of colors. InCEA introduces a different crossover operation and an optimized local search process. The InCEA algorithm’s basic approach can be shown as follows:

4.1.1 Individual Representation and Initial Population Generation

In InCEA, each individual represents a possible solution composed of color classes, where vertices within a class do not conflict. The initial population is generated using metric-based rules that prioritize diverse assignments to achieve a range of solutions from the start.

4.1.2 Integrated Crossover Operator (InCX)

The InCX operator in InCEA combines color classes from two parent solutions while handling conflicts using a conflict pool and a search-back operation. It randomly selects color classes from both parents to form an offspring with the largest group of non-conflicting vertices for each color class. Vertices that can't be assigned without conflicts are put into a pool to be checked later. In the search-back operation, the algorithm tries to fit these unassigned vertices into existing classes in the offspring. If this isn't possible, the vertices stay in the pool.

4.1.3 Local Search with Weighted-Swap (W-SWAP)

InCEA then applies a local search to further optimize each solution, aiming to minimize the weight sum of uncolored vertices. The W-SWAP operation achieves this by evaluating vertex swaps based on their weights. The algorithm sorts vertices in the pool by weight, attempting to replace higher-weight uncolored vertices with lower-weight conflicting vertices from the color classes. This approach prioritizes minimizing the total uncolored weight, enhancing the fitness of the solution.

The InCEA paper presents a relevant approach to solving graph coloring using evolutionary algorithms and local search techniques, similar to our project. However, while InCEA focuses on vertex-weighted graphs, our project targets large-scale unweighted graphs. This difference requires us to design a different crossover operation and adapt the algorithm accordingly. We can draw inspiration from InCEA's methods, particularly its crossover and local search operations, but we will modify them to suit our specific problem of handling large graphs (100k nodes) using bitwise operations and parallelization for scalability.

4.2 BitEA: BitVertex Evolutionary Algorithm to Enhance Performance for Register Allocation [5]

BitEA: BitVertex Evolutionary Algorithm to Enhance Performance for Register Allocation study published in 2024 by Gizem Sungu Terci, Enes Abdulhalik, Alp Arslan Bayrakcı and Betul Boz aims to improve the design and effectiveness of the InCEA evolutionary algorithm. The InCEA evolutionary algorithm keeps the graph as an integer. Unlike traditional methods, the BitEA evolutionary algorithm uses a structure that represents each node as a bit. This bit-based approach reduces memory usage by up to four times. Also, BitEA reduces the processing time by using bit operations during color assignments. BitEA working process is shown below:

4.2.1 Graph Representation to Bit Level and Initial Population

In the algorithm, each node is represented as a bit. For example, for a 32-node graph, each node is encoded as a bit in a 32-bit integer. Multiple bit sequences are used to represent graphs larger than 32 nodes. An initial population is created randomly or according to certain criteria. Each individual represents a graph coloring solution and the nodes are assigned colors for each individual.

4.2.3 Crossover Operation, Improved Search Back (ISB) and Local Search

In the operator called BitCX, one of the color classes of both parents is randomly selected and the nodes in these color classes are combined with the bitwise OR operation and transferred to the new individual. If there is a conflict in the new individual, the nodes creating the conflict are transferred to the pool and these nodes are removed from the color class with the bitwise XOR operation.

After the BitCX operation, if there are nodes added to the pool, ISB takes each node from the pool and checks if there is any conflict in the previous color classes and replaces it with the least weighted conflicting node.

A local search operation called Bit-SWAP is performed for nodes added to the pool and could not be placed in the ISB operation. In this process, a color class is tried to be found that minimizes the conflicts for each node. If the conflict in a color class is at a minimum level, the conflicting nodes are taken into the pool and the relevant node is assigned to this color class.

4.2.6 Fitness Function, Population Update and Loop

The fitness value for the child is calculated. Fitness is measured by the total weight of the nodes that cannot be assigned to the solution. If the fitness of the new individual is better than its parents, it is replaced with one of the parents in the population. After the fitness function, the population is updated. The algorithm repeats until the specified iteration or time limit is reached. During this time, BitEA tries to find the most suitable solution in the population.

BitEA work is based on keeping graphs as bits and performing the operations of the evolutionary algorithm bitwise. However, this project has fundamental differences from BitEA work. Since large-scale graphs will be used in this project, it is necessary to use a large number of 32-bit integers to keep the entire graph as bits, so the complexity may increase. The BitEA algorithm is designed for graphs with weighted edges, and such graphs are out of scope in this project. Another difference is that since the evolutionary algorithm will be different, bitwise operations will also be different.

4.3 BitColor: Accelerating Large-Scale Graph Coloring on FPGA with Parallel Bit-Wise Engines [6]

This study aims to enhance large-scale graph coloring by leveraging an FPGA with several key techniques: bitwise processing, a high-degree vertex cache (HVC), a graph pruning strategy to reduce off-chip memory access, a data conflict table, and a multi-port HVC to enable simultaneous vertex coloring. The primary implementations are as follows:

4.3.1 Bitwise Representation and Processing

Colors are represented by binary strings, where each bit indicates whether a color is available. To determine a vertex's color state, bitwise OR operations are applied between color strings of neighboring vertices. After this, bitwise AND and NOT operations are used to select a color based on a greedy algorithm. To address memory constraints, colors are compressed and decompressed using custom hardware.

4.3.2 High-Degree Vertex Cache (HVC) and Graph Prune Strategy

High-degree vertices are cached in HVCs, as these vertices have numerous neighbors and are frequently accessed. This caching reduces memory access times. To handle read conflicts across multiple compute units, the HVC is implemented as a multi-port cache. The graph prune strategy further reduces DRAM access by removing uncolored neighbors, streamlining both memory use and computation.

4.3.2 Data Conflict Table

A custom hardware-level table resolves conflicts that arise when different compute units attempt to color the same vertex. Each row in this table records a compute unit, the vertex being colored, a bit indicating coloring completion, the selected color, and a conflict status bit. If a conflict occurs, the compute unit with the lower index completes its coloring first.

In contrast to our approach, this study uses a bitwise greedy algorithm and hardware-level solutions to tackle large-scale graph coloring. Our work, however, focuses on an evolutionary algorithm and software-based solutions, including graph reduction techniques. Similar to our approach, this study employs a bit representation of colors, bitwise operations, and parallelization. However, unlike their specialized hardware-driven parallelization, our approach uses general-purpose parallelization with a bitwise representation of the graph.

5. SCOPE

This project focuses on developing an optimized solution for the graph coloring problem specifically targeting large graphs. Our scope includes working with graphs that have an edge density of less than 0.8, as fully connected graphs are excluded due to the minimal relevance of coloring in such dense structures. We limit our approach to undirected and unweighted graphs, excluding directed and weighted graphs from our considerations. Additionally, this project will address static graphs only, as dynamic graph structures are outside its scope.

For this project, large graphs are defined as those containing at least 100,000 nodes, with our methods emphasizing efficient memory usage and computational performance at this scale. We assume that excluding less connected nodes will contribute to optimized memory utilization. The hardware setup for this project includes a 24-core CPU (such as an AMD Threadripper) and 128 GB of RAM, which we assume will be enough to meet the memory and processing requirements for the tasks at hand.

In addition to hardware constraints, there are some software constraints such as using GCC compiler to build our project and the project must successfully compile. Also, we are planning to use OpenMP for parallelization and CUDA for GPU implementation.

6. METHODOLOGY and TECHNICAL APPROACH

To address the graph coloring problem for large-scale graphs, we propose a scalable and efficient approach that takes advantage of evolutionary algorithms, inspired by InCEA, alongside bitwise representations and operations from BitEA [5] and BitVertex [6] paper’s techniques. This approach is designed to handle the complex demands of coloring graphs that have 100,000 nodes while minimizing memory and computational requirements.

Our method begins with Graph Reduction, selectively removing lower-degree nodes to simplify the graph, making it more manageable and focusing on densely connected areas. We then use Bitwise Representation to efficiently encode graph connections and color classes, significantly reducing memory usage and enhancing processing speed. The core of our approach is an Evolutionary Algorithm, where potential coloring solutions are iteratively improved through selection, crossover, and mutation. This process is enhanced with Parallelization via OpenMP to further accelerate solution finding.

The flow of our proposed approach is given in Figure 1. By combining these methods, we aim to introduce a solution that efficiently colors large-scale graphs, potentially reducing the chromatic number with a computationally optimized and scalable solution.

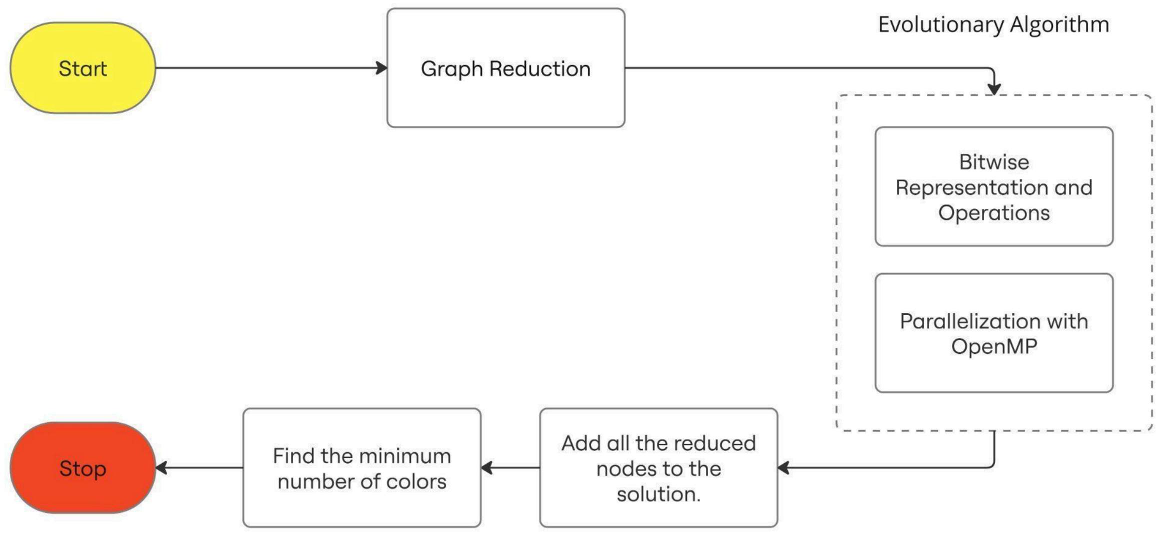
Start
Graph Reduction
Evolutionary Algorithm
Bitwise Representation and Operations
Parallelization with OpenMP
Add all the reduced nodes to the solution.
Find the minimum number of colors
Stop

Figure 1: Flowchart of the Proposed Approach for the Graph Coloring Problem

6.1 Graph Reduction

Our approach includes a graph reduction technique designed to simplify large graphs by selectively removing nodes with fewer connections. By removing these lower-degree nodes, we can effectively reduce the graph's complexity, making it smaller and easier to process while retaining the structure of its more interconnected areas. This reduction will not only decrease the computational load but also improve the efficiency of our coloring algorithm. Once the coloring of the reduced graph is complete, we will reintroduce the removed nodes, reintegrating them in a way that maintains the integrity of the final colored graph. This approach allows us to handle large-scale graphs more effectively by focusing computational resources on the most connected nodes first and addressing less connected areas afterward.

6.2 Graph Representation

The bit representation of graphs uses less memory compared to traditional integer-based representations. In this representation, a separate 32-bit integer is used for the connections of each node. For example, if node n0 has connections to n5 and n12, the 5th and 12th bits in the 32-bit integer of node n0 are set to 1, and the remaining bits are left as 0. If the graph has more than 32 nodes, more than one 32-bit integer is used for each node. For example, if node n0 has connections to n6 and n36, in the 32-bit integer set used for the bit representation of node n0, the 6th bit in the first set and the 4th bit in the second set are marked as 1, and the remaining bits are 0.

The bit representation of colored graphs, a separate 32-bit integer is used for each color class. For example, if nodes n0 and n15 are the same color, the 0th and 15th bits in the 32-bit integer representing this color are set to 1, and the remaining bits are left as 0. If the number of nodes is more than 32, more than one 32-bit integer is used for each color class. For example, if n10 and n40 are painted in the same color, the 10th bit in the first integer and the 8th bit in the second integer are marked as 1 in the 32-bit integer set used for the bit representation of this color, and the remaining bits remain as 0. This method allows the graph's color classes to be represented efficiently in memory.

6.3 Evolutionary algorithm

The evolutionary algorithm aims to find the best among the solutions by taking as a model the natural selection and genetic transmission mechanisms in the biological evolution process. Usually, a randomly created population is used at the beginning and each individual in this population is considered as a potential solution. You can see the general flow of an evolutionary algorithm in Figure 2 below.

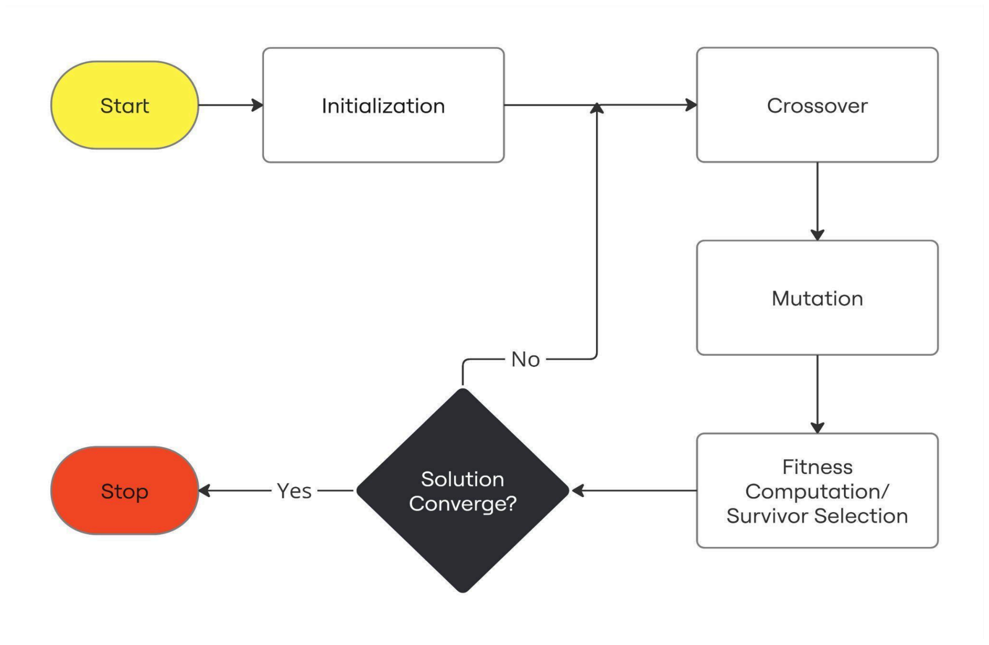

Figure 2: General flowchart for Evolutionary Algorithm

The first stage of the algorithm is to create the population randomly or within the framework of certain rules. Each individual is created with various coloring combinations of nodes and has different genetics, i.e. solutions. The most successful individuals are selected from the population by considering the number of neighboring nodes with the same color and the total number of colors used for the individuals to be used in the next generation.

After the selection process, the solutions of the selected individuals are brought together by the process called crossover and new individuals with better performance are produced. In order for the new individuals to add more diversity to the population, the mutation process is performed by randomly changing the colors of the nodes. After the mutation, the number of neighboring nodes with the same color is reduced by making small changes in the coloring with local search. Finally, when a certain number of generations is reached or the solution is sufficiently optimized, the algorithm is terminated and the best solution is selected according to the quality of the individuals in the population.

6.4 Graph Recombination

The graph recombination phase is designed to restore the graph's integrity and complete the coloring of any uncolored nodes. In this phase, nodes removed during graph reduction are reintroduced, and each re-added node is assigned an appropriate color based on the colors of its neighboring nodes. Because these nodes were initially removed due to their few connections, they can be colored efficiently and without conflict. This process produces the final solution, ensuring a complete and consistent coloring of the entire graph.

6.5 Performance Evaluation

This project aims to advance the capabilities of graph coloring algorithms, pushing the boundaries of state-of-the-art methods and contributing valuable improvements to fields that depend on large-scale graph optimization. In order to achieve this, we will have three implementations, ***Base Implementation*** which targets to solve graph coloring problem using an evolutionary algorithm, ***Bitwise Implementation*** which will use bitwise operations to decrease memory usage and increase performance and finally ***Bitwise-Parallel Implementation*** that is going to add parallelization to Bitwise Implementation.

Table 1: Table of large graphs to be used in testing

Graphs	Nodes	Edges	Categories
ego-Facebook (EF) [8]	4.1K	88.2K	Social network
gemsec-Deezer_HR (GD) [9]	54.5K	498.2K	Social network
com-DBLP (CD) [10]	317.0K	1.0M	Collaboration network
com-Amazon (CA) [10]	335.8K	925K	Product network

In the first set of tests, to check the outputs of the programs and test the implementations, synthetically generated benchmarks with at most 500 nodes will be used. Using these benchmarks, we will be comparing the performance of the three proposed implementations. Once the outputs are validated, we will proceed with tests including large graphs. These graphs with their properties are given in Table 1. These large graphs will be used to compare the proposed method with the state-of-the art algorithms such as BitColor [6]. GCC and OpenMP are utilized as software resources whilst computational facilities are employed for execution.

7. PROFESSIONAL CONSIDERATION

7.1 Methodological considerations/engineering standards

This project complies with various methodological and engineering standards for systematic development. The following standards and tools will be used throughout the project:

Algorithm Implementation: The algorithms are implemented using the C programming language. C ensures high efficiency, performance and fast execution, and provides control over the system's resources. Thus, it supports the creation of robust algorithms that effectively process large-scale data structures.

Version Control: Source code management is done using Git, a version control system that is recognized and used on platforms like GitHub. This control system facilitates team collaboration and allows the creation of versions of the project.

Project Management: GANTT charts are used to visualize the project schedule and monitor progress across different stages. This tool helps plan timelines, identify dependencies, and track milestones. Unified Modeling Language (UML) diagrams are used to model system architecture and design. These diagrams facilitate communication between team members.

IEEE Standards: The project is based on IEEE standards for software engineering and documentation. This includes following guidelines for software requirements specifications, design documentation, and testing procedures, and providing a structured approach to software development.

7.2 Realistic Constraints

In the design and execution of this project, several realistic constraints are considered to ensure that the solution is viable, ethical, and beneficial to society. These constraints include:

Economical: The focus of the project is on minimizing expenses associated with software development, testing, and infrastructure by prioritizing the use of open-source and free resources whenever possible.

Environmental: Strategies are implemented to optimize resource usage, thus minimizing the carbon footprint associated with large-scale computations.

Ethical: Ethical considerations related to data usage and algorithm fairness are addressed. The project ensures that the data used does not violate privacy standards.

Health and Safety: The project does not pose any health or safety risks.

Sustainability: The sustainability of the solutions developed is emphasized. The project aims to create algorithms that can efficiently solve large graph problems with minimal resource consumption, contributing to long-term sustainability in computational methods.

Social: There are no elements of social interaction involved in the project.

7.3 Legal Considerations

There is no legal consideration. The tools and data to be used in the project are open source and do not require permission or license.

8. MANAGEMENT PLAN

The management of this project has different phases, each with specific tasks and durations. Below is a description and timeline of each task phase and highlights key milestones throughout the project lifecycle.

8.1 Description of all task phases and their durations.

Phase 1: Literature Review (6 Weeks): Conduct a comprehensive review of existing literature on graph coloring problems and optimization techniques to gather insights and identify relevant studies.

Phase 2: Dataset and Benchmarking (4 Weeks): Source a dataset of large graphs suitable for testing. This phase includes finding benchmark results for graph coloring to evaluate the performance of the developed algorithms.

Phase 3: Algorithm Design (8 Weeks): Design the structure of the evolutionary algorithm for node coloring, defining parameters and optimization strategies to enhance efficiency.

Phase 4: Large Graph Reduction (4 Weeks): Implement graph reduction techniques to simplify large graphs while preserving essential characteristics, thereby facilitating faster processing in subsequent phases.

Phase 5: Implementation of Evolutionary Algorithm (6 Weeks): Execute the core implementation of the evolutionary algorithm, integrating graph reduction techniques and thoroughly testing for accuracy and functionality.

Phase 6: Implementation of Parallelization (5 Weeks): Develop the parallelization of the graph coloring process, optimizing the algorithm to run concurrently on multiple threads or processors to reduce computation time.

Phase 7: Testing on Dataset (6 Weeks): Conduct rigorous testing of the developed algorithm on the created dataset of large graphs, evaluating performance based on previously collected benchmarks.

Phase 8: Final Adjustments and Documentation (2 Weeks): Make final adjustments based on testing results, document the entire project, and prepare for the final presentation, including compiling findings, results, and methodologies.

8.2 Division of responsibilities and duties among team members.

The division of responsibility among team members in each phase is given as percentages in Figure 3 below.

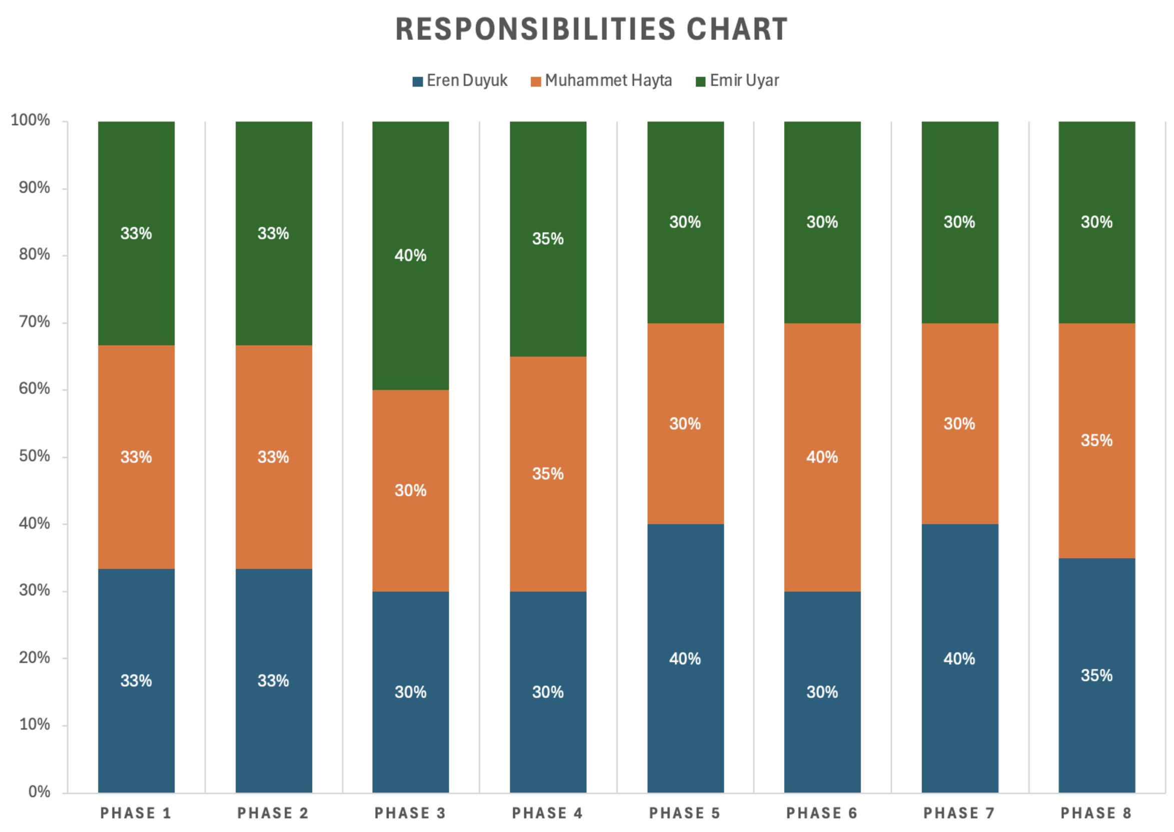

Figure 3: Chart illustrating the distribution of responsibilities

8.3 Timeline with milestones

The project timeline, including the duration of each phase and milestones marked with flags, is shown in Figure 4 below.

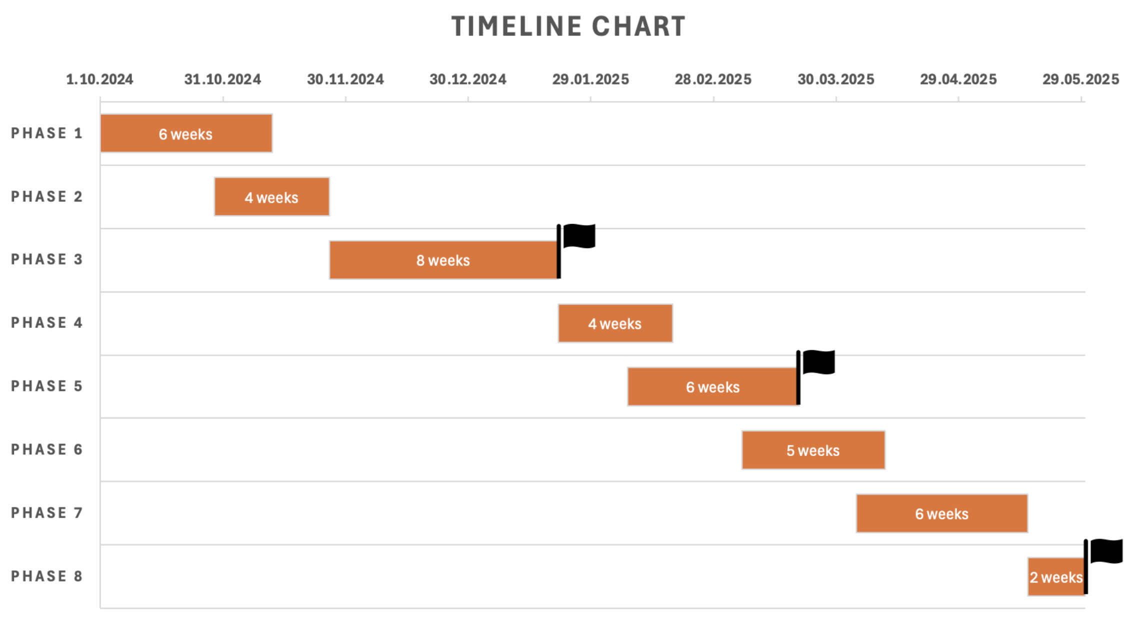

Figure 4: Chart illustrating the timeline distribution

Milestone 1: Completion of literature review and algorithm design.

Milestone 2: Completing the implementation of the project.

Milestone 3: End of the project.

9. SUCCESS FACTORS and RISK MANAGEMENT

9.1 Measurability/Measuring Success

(i) Objective 1: Design an evolutionary algorithm that can consistently find solutions close to the best possible coloring for large graphs. The algorithm should effectively explore different coloring possibilities, creating a reliable tool that works well across various large graph types.

Success Factor: The algorithm should yield solutions that use, on average, no more than 20% additional colors compared to the optimal solution.

(ii) Objective 2: Improve the algorithm's efficiency by optimizing memory usage and reducing runtime, allowing it to handle large graphs. This includes using bitwise operations, parallel processing, and efficient data structures to lower resource demands while keeping performance high.

Success Factor: These optimizations, on average, should reduce memory usage by at least 30% and improve execution time by at least 85% compared to the original algorithm. The memory reduction will mainly be achieved through the use of bits to represent nodes and connections, while the improvement in execution time will result from bitwise operations, parallel processing, and graph reduction techniques. The expected execution time improvement can be represented by the formula:

$$Execution\ Time_{new} = \frac{Execution\ Time_{old}}{(1+0.85)}$$

(ii) Objective 3: Conduct extensive benchmarking and testing on a diverse set of large graph datasets to thoroughly evaluate the algorithm's scalability and performance. Testing across various graph types and sizes will ensure that the algorithm performs at least comparably to existing algorithms in the literature, both in terms of execution efficiency and solution quality.

Success Factor: Testing will be based on the procedures outlined in [7]. Scalability will refer to the algorithm’s ability to handle increasingly larger graphs while maintaining a linear increase in execution time. Performance will be measured by the quality of the results, assessing how close they are to the optimal solutions. The goal is to surpass the state of the art algorithms by at least 20% in terms of solution quality, 30% in execution time, and 5% in memory efficiency using Snap datasets [11].

9.2 Risk Management

Risk 1: Potential failure in effectively reducing graph size to a manageable level through graph reduction techniques.

Resolution: If challenges arise, graph reduction techniques may be set aside in favor of focusing on medium-sized graphs, limiting the size to 10,000 nodes and an edge density below 0.7.

Risk 2: Insufficient resources to process large graphs as planned.

Resolution: Should resource limitations be encountered, we may use cloud-based computing resources such as Google Collab or Hetzner, which offer high computing power to support large-scale processing.

Risk 3: Difficulty in efficiently processing large graphs using OpenMP to yield results within a feasible time frame.

Resolution: If OpenMP fails to deliver adequate speed, GPU-level parallelization may be implemented using libraries such as CUDA to leverage GPU processing power for improved computational performance.

10. BENEFITS and IMPACT of the PROJECT

Our project aims to develop a scalable and efficient evolutionary algorithm for large-scale graph coloring, capable of handling graphs with up to 100,000 nodes. This innovative approach offers valuable benefits for both the scientific community and industries where graph optimization is essential. The algorithm's scalability and efficiency could attract interest from major tech companies like Google and Facebook, given their extensive work with large-scale graph data.

10.1 Scientific Impact

Our project has the potential to advance the field of graph theory by introducing efficient, large-scale graph coloring techniques that are both scalable and optimized for computational performance. This project has a high likelihood of resulting in publishable research that could contribute to scientific literature on graph optimization, especially if successful in handling large graphs effectively.

10.2 Economic/Commercial/Social Impact

Optimizing graph coloring has significant applications in areas such as network management, resource allocation, and scheduling, where efficient solutions can lead to cost savings and better resource management. A successful approach to large-scale graph coloring could also be developed into a commercial software or tool, benefitting industries with complex, large-scale scheduling and network management requirements. Additionally, improving computational efficiency through bitwise and parallel methods contributes to energy sustainability by reducing the energy consumption required for large computational tasks.

10.3 Potential Impact on New Projects

This project has the potential to act as a foundation for future research in large-scale graph algorithms. Future projects could build on our work by extending it to dynamic graphs or adapting the optimized techniques to other graph-related problems. Also, the development of a scalable solution also could open opportunities for interdisciplinary applications in fields that utilize large graphs, such as bioinformatics, social network analysis, and logistics.

10.4 Impact on National Security

Efficient graph coloring solutions have applications in national security, particularly in areas like cybersecurity and resource management. In cybersecurity, graph coloring algorithms are useful for optimizing resource allocation in threat detection systems and network traffic management. Additionally, optimized graph algorithms can enhance the reliability of communication networks, which are critical for emergency response and border security operations.

***This document was collaboratively prepared by the team members, with ChatGPT employed to refine the quality of the English language used throughout.

REFERENCES

[1] Garey, M. R., Johnson, D. S. 1990. Computers and Intractability: A Guide to the Theory of NP-Completeness. New York, NY, USA: Freeman.

[2] Jensen, T. R., Toft, B. 1995. Graph Coloring Problems. New York, NY, USA: Wiley.

[3] Lewis, R. M. R. 2015. A Guide to Graph Colouring, Berlin, Germany: Springer.

[4] Quong, R. W., Chen, S.-C. 1993 “Register allocation via weighted graph coloring”, Purdue Univ. Lib., USA, ECE Tech. Rep. TR-EE 93-23 (232).

[5] Terci, G. S., Abdulhalik, E., Bayrakcı, A. A., Boz, B. 2024. “BitEA: BitVertex Evolutionary Algorithm to Enhance Performance for Register Allocation”, IEEE Access, 12, 115497-115514.

[6] Fan, H., Li, M., Wu, J., Lu, W., Li, X., Yan, G. 2023. ‘‘BitColor: Accelerating large-scale graph coloring on FPGA with parallel bit-wise engines”, in Proc. 52nd Int. Conf. Parallel Process., 492–502.

[7] Boz, B., Süngü, G. 2020. ‘‘Integrated crossover based evolutionary algorithm for coloring vertex-weighted graphs’’, IEEE Access, 8,126743–126759.

[8] Leskovec, J., Mcauley J. 2012 “Learning to discover social circles in ego networks”, Advances in neural information processing systems, 25.

[9] Rozemberczki, B., Davies, R., Sarkar, R., Sutton, C. 2019. “GEM-SEC: Graph Embedding with Self Clustering”, In Proceedings of IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ACM, 65–72.

[10] Yang, J., Leskovec, J. 2012. “Defining and evaluating network communities based on ground-truth”, In Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics. 1–8.

[11] Leskovec, J., Krevl, A. “Snap datasets: Stanford large network dataset collection”, http://snap.stanford.edu/data Date accessed: 13 Kasım 2024.