

**SHRI RAMDEOBABA COLLEGE OF ENGINEERING AND MANAGEMENT**

**Department of Computer Science & Engineering**

**Session: 2023-24**

**DESIGN AND ANALYSIS OF ALGORITHMS**

**PROJECT**

**V Semester, B.E.**

**Group Members**

**Maitri Markandeywar (07)**

**Niharika Kumar (19)**

**Prerna Chourasia (10)**

**Rishab Jain (54)**

**Course Co-ordinators**

**Dr. S. Hira**

**Dr. K. Khurana**

**Deep Learning-based Hybrid Graph-Coloring Algorithm**

**for Register Allocation**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Serial No.** | **Topic** | **Page No.** |
| 1. | Problem Definition | 4 |
| 2. | Methodology | 5 |
| 3. | Algorithm | 6 |
| 4. | Output | 8 |
| 5. | Discussion on Time Complexity | 9 |
| 6. | Diagrams | 10 |
| 7. | Conclusion | 12 |

**PROBLEM DEFINITION**

Modern compiler optimization faces the intricate challenge of efficiently assigning registers to variables in a program, balancing the constraints imposed by data dependencies and the limited availability of registers. This problem is formally defined as the Graph Coloring Problem, an NP-hard task with broad implications for the performance of compiled code. Each node in the graph corresponds to a variable in the program. Edges between nodes signify data dependencies, dictating that variables connected by an edge cannot share the same register.

Colours represent registers, and the objective is to assign colours to nodes in a manner that avoids assigning the same color to adjacent nodes (variables with dependencies). A valid colouring is an assignment that adheres to the no-adjacent-nodes-same-color constraint. Efficiently allocate registers to minimize the number of registers used while respecting data dependencies, thus optimizing the program's execution on a processor.

Graph coloring is recognized as NP-hard, underscoring the inherent difficulty in finding optimal solutions for larger programs.Optimal register allocation directly impacts the runtime performance and memory efficiency of compiled code.This problem sits at the crossroads of graph theory, compiler design, and optimization, requiring innovative solutions for practical implementation.

**METHODOLOGY**

The authors propose a hybrid algorithm to address the Graph Coloring Problem for register allocation. The methodology begins with a deep learning (DL) network, specifically designed with Long Short-Term Memory (LSTM) layers, to allocate colors to nodes in the interference graph. However, the authors acknowledge a limitation in the DL approach: the potential for the DL network to allocate the same color to nodes connected by an edge, leading to an invalid coloring.

To overcome this challenge, the authors introduce a color correction phase that follows the DL network allocation. This phase aims to rectify invalid colorings by identifying edges with shared colors and subsequently applying corrective measures. The color correction algorithm operates on the premise that reusing colors allocated to unrelated nodes can resolve invalid colorings. If such color reuse is not feasible, the algorithm introduces a completely new color, thereby breaking the invalid allocation.

The DL network is trained using a diverse dataset comprising several thousand random graphs with varying sparsity. This dataset serves as the input for the DL network, allowing it to learn intricate patterns and relationships within interference graphs. The training dataset includes labels indicating the validity of the colorings on edges, assisting the DL network in discerning valid and invalid color allocations.

The authors also present pseudocode for the color correction phase, detailing the step-by-step process of identifying invalid edges, attempting color reuse, and, if necessary, introducing new colors to ensure a valid coloring of the interference graph.

In evaluating their hybrid approach, the authors discuss their experiences with various graphs, noting that approximately 10% to 30% of edges may initially receive an invalid coloring from the DL network. The results are likely assessed based on metrics such as the percentage of corrected edges, overall graph coloring quality, and computational efficiency.

This integrated methodology combines the strengths of deep learning for capturing complex patterns in interference graphs with a subsequent color correction phase to ensure the validity of the color allocations. The training on diverse datasets and the systematic approach to correcting invalid colorings contribute to the robustness of the proposed hybrid algorithm.

**ALGORITHM**

*algorithm ColorCorrection()*

*Colors = Set of colors allocated to the graph after inference*

*for each INVALID edge e=<n1,n2> do {*

*Let c=color of the nodes n1 and n2*

*for (c1 in Colors AND c1 != c) do {*

*if (exists e1 = <n1,M> such that M has color c1)*

*continue;*

*Color c1 is not used by any neighbor of n1*

*reuse color c1 for n1*

*}*

*if no color found for reuse for n1 then {*

*for (c2 in Colors AND c2 != c) do {*

*if exists e1 = <n2,M> such that M has color c2*

*continue;*

*Color c2 is not used by any neighbor of n1*

*reuse color c2 for n1*

*}*

*}*

*if no color found for reuse for both n1 AND n2 then {*

*Create new color cn*

*Assign cn to n1*

*Colors = Colors U {cn}*

*}*

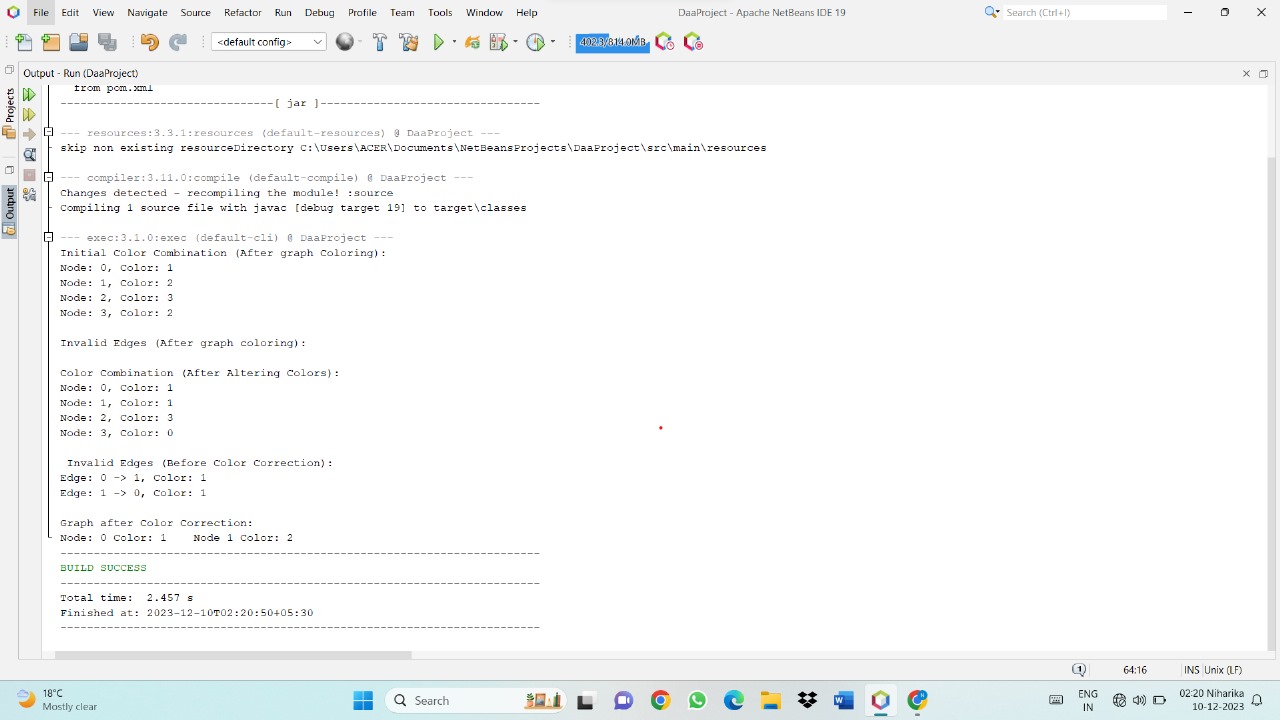
*}*

Here is a more detailed explanation of each step in the algorithm:

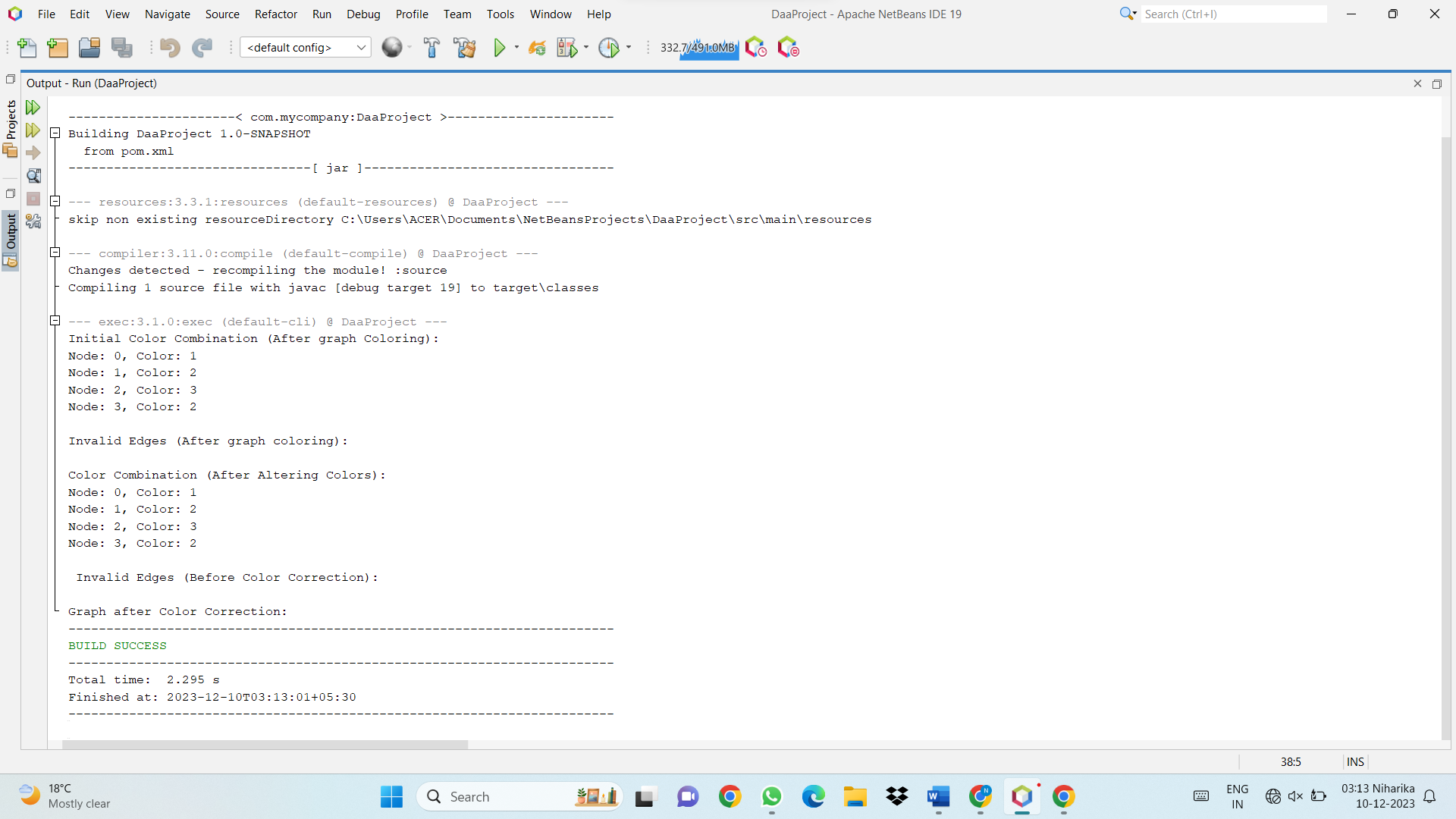
1. Initialize the set of colors Colors to the set of all possible colors.
2. Iterate over all invalid edges e in the graph
3. Let c be the color of the nodes n1 and n2 connected by edge e.
4. Iterate over all colors c1 in the set Colors that are different from c.
5. If there exists an edge e1 between node n1 and another node M such that M has color c1, then skip to the next color.
6. Otherwise, c1 is not used by any neighbor of n1, so assign c1 to n1 and break the loop.
7. If no color could be found for reuse for n1, then iterate over all colors c2 in the set Colors that are different from c.
8. If there exists an edge e1 between node n2 and another node M such that M has color c2, then skip to the next color.
9. Otherwise, c2 is not used by any neighbor of n2, so assign c2 to n1 and break the loop.
10. If no color could be found for reuse for either n1 or n2, then create a new color cn and assign it to n1.
11. Add cn to the set Colors.

**OUTPUT**

1.When there are invalid edges



2.When there are no invalid edges

****

**DISCUSSION ON TIME COMPLEXITY**

The color correction phase in the given code has a worst-case time complexity of O (E \* N^2), where E is the number of edges in the graph and N is the number of nodes in the graph. Let's break down the complexity analysis:

1. Iteration over Edges (E)

The code iterates over each edge in the `invalidEdges` set once.

2. Nested Iteration over Nodes (N)

For each edge, there is a nested iteration over nodes. In the worst case, for each node, the code iterates over all colors, checking if they are available for both start and end nodes.

3. Color Assignment

The color assignment involves finding available colors for each node and assigning a color. In the worst case, all nodes may need to be reassigned colors, resulting in additional iterations over colors.

4.Set Operations

The code involves set operations, such as checking if a node or color is in a set.

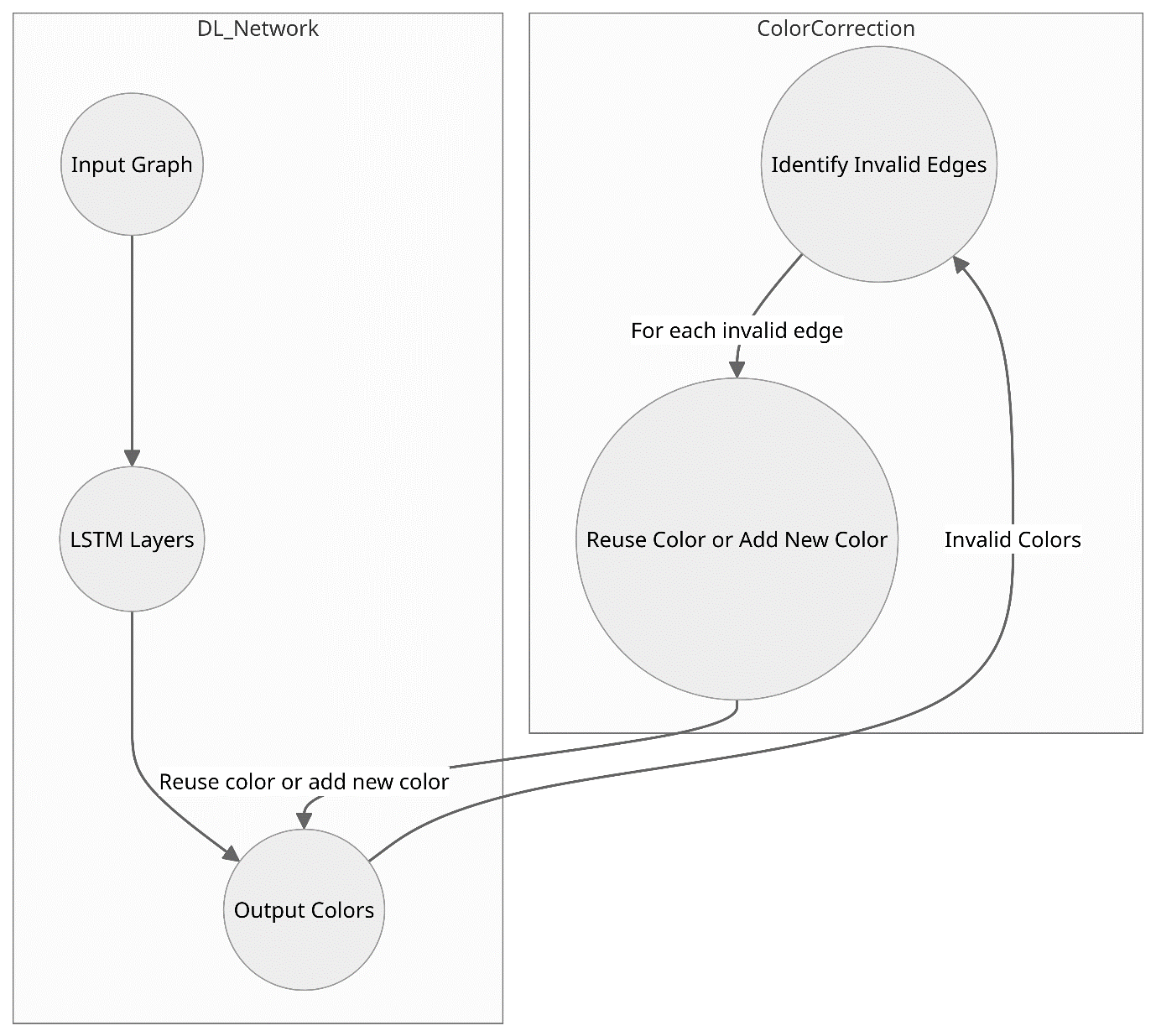
Combining these factors, the worst-case time complexity is O (E \* N^2). The nested iteration over nodes and colors contributes to the N^2 factor and iterating over edges contributes to the E factor.

It's important to note that this worst-case complexity may occur when all nodes need color correction and when there is a high degree of interference between nodes, leading to a significant number of iterations over colors for each node.

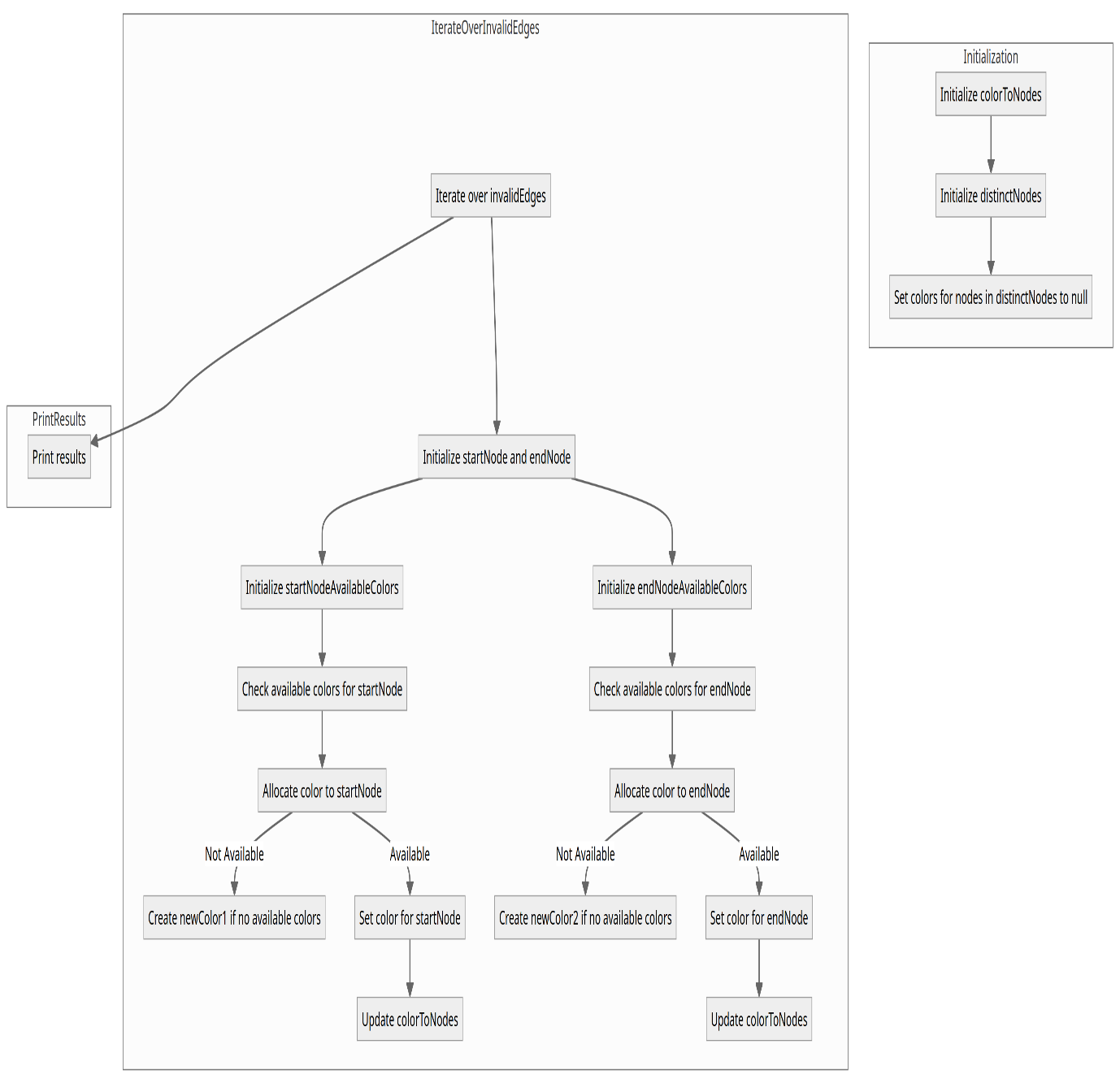
However, the average-case complexity may be lower, especially if not all nodes require color correction, and the interference between nodes is limited. The actual performance will depend on the characteristics of the input graph.

**DIAGRAMS**

1. **Deep Learning-based Hybrid Graph-Coloring Algorithm for Register Allocation**

****

1. **Color Correction Phase**

****

**CONCLUSION**

In summary, the presented methodology combines the power of deep learning with a targeted color correction algorithm to address the Graph Coloring Problem for register allocation. The deep learning network, leveraging LSTM layers, demonstrates its ability to allocate colors to nodes based on the complex patterns and relationships within the interference graph. However, recognizing the potential for invalid colorings, the authors introduce a color correction phase. This additional step systematically rectifies invalid colorings by reusing colors from unrelated nodes or introducing entirely new colors.

The color correction algorithm, outlined in pseudocode, iterates over invalid edges, attempting to resolve conflicts and ensuring a valid assignment of registers to variables. The success of the hybrid approach is further emphasized through the authors' experiences with diverse graphs, where approximately 10% to 30% of edges initially receive invalid colorings.

The integration of deep learning and the color correction phase presents a holistic solution to the Graph Coloring Problem for register allocation. The hybrid algorithm not only harnesses the capacity of deep learning to capture intricate graph patterns but also provides a robust mechanism to address potential shortcomings, resulting in an efficient and valid register allocation scheme. This approach is poised to contribute significantly to the optimization of compiler performance and the generation of high-quality machine code.