COLORING GRAPHS WITH INTERVALS FOR PARALLEL COMPUTING

by

Dante Durrman

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Applied Mathematics

Charlotte

2024

Approved by:

____
Dr. Erik Saule

____
Dr. Gabor Hetyei

____
Dr. Evan Houston

____
Dr. Min Shin

©2024
Dante Durrman
ALL RIGHTS RESERVED

ABSTRACT

DANTE DURRMAN. Coloring graphs with intervals for parallel computing. (Under the direction of DR. ERIK SAULE)

Graph coloring is commonly used to schedule computations on parallel systems. Given a good estimation of the computational requirement for each task, one can refine the model by adding a weight to each vertex. Instead of coloring each vertex with a single color, the problem is to color each vertex with an interval of colors.

We study this problem for particular classes of graphs, namely stencil graphs. Stencil graphs appear naturally in the parallelization of applications, where the location of an object in a space affects the state of neighboring objects. Rectilinear decompositions of a space generate conflict graphs that are 9-pt stencils for 2D problems and 27-pt stencils for 3D problems.

We show that the 5-pt stencil and 7-pt stencil relaxations of the problem can be solved in polynomial time. We prove that the decision problem on 27-pt stencil is NP-Complete. We discuss approximation algorithms with a ratio of 2 for the 9-pt stencil case, and 4 for the 27-pt stencil case. We identify two lower bounds for the problem that are used to design heuristics. We evaluate the effectiveness of several different algorithms experimentally on a set of real instances. Furthermore, these algorithms are integrated into a real application to demonstrate the soundness of the approach.

Executing graph algorithms in a parallel or distributed context is a challenging problem. Solving race conditions with locks is usually prohibitively expensive and some algorithms opt for a strategy that ignores the race condition altogether and corrects later the derived solution if it is invalid. Alternatively, dataflow algorithms solve the synchronization problem by executing the algorithm by following a partial order on the graph. While removing the cost of locks or avoiding a checking phase

improves performance, it is possible that the algorithm picks a partial order with long chains, which limits its utility to parallel applications.

We investigate how distributed dataflow graph algorithms obtain a partial order and how one could favor orders with shorter long chains. Most dataflow algorithms obtain their order by having each vertex of the graph pick a uniformly random number in $[0, 1)$ and order the vertices based on that number. We believe that this type of order could lead to long chains in graphs with dense regions such as small world graphs. We design two alternative ways of generating the order to make it similar to a largest degree first order. We study the behavior of these different algorithms on a wide range of randomly generated RMAT graphs and on a set of real-world graphs. We show that our ordering methods can significantly reduce the length of the longest chain.

DEDICATION

Thank you to my mom who has supported me through my entire academic career and allowed me to pursue this opportunity.

TABLE OF CONTENTS

LIST OF TABLES ix

LIST OF FIGURES x

LIST OF ABBREVIATIONS 1

CHAPTER 1: INTRODUCTION 1

1.1. Problem Statement 4

1.2. Document Structure 5

CHAPTER 2: PRELIMINARIES 7

2.1. Basic Definitions and Theorems 7

2.2. Related Works 11

CHAPTER 3: INTERVAL COLORING WITH STENCILS 14

3.1. Introduction 14

3.2. Interval Coloring Problem of Stencils 15

3.2.1. Problem Definition 15

3.3. Special Case Analysis 17

3.3.1. Cliques 17

3.3.2. Bipartite Graph 17

3.3.3. Odd Cycles 18

3.3.4. Lower bounds are not tight 21

3.4. NP-Completeness 22

3.5. Heuristics 27

3.5.1. Greedy Algorithms 27

3.5.2. Bipartite Decomposition 29

3.6. Experiments 30

3.6.1. Experimental Setting 30

3.6.2. 2D Results 31

3.6.3. 3D Results 33

3.6.4. Optimal coloring based analysis 35

3.7. Coloring for Space Time Kernel Density Estimation 36

3.8. Conclusion 39

CHAPTER 4: DATAFLOW ALGORITHMS 43

4.1. Introduction 43

4.2. Problem Statement 44

4.2.1. Dataflow Algorithms 44

4.2.2. Combinatorial Optimization Model 45

4.3. Deriving better partial orders 47

4.3.1. Methods 47

4.3.2. Basic analysis 48

4.4. Study on Recursive Graph Model (RMAT) 49

4.4.1. Methodology 49

4.4.2. Initial Investigation 51

4.4.3. Exploring the RMAT Parameter Space 52

4.5. Real Graph Study 53

4.6. Conclusion 54

CHAPTER 5: LONG PATHS IN ORIENTED $G_{N,P}$ GRAPHS 58

5.1. Introduction 58

5.2. Long Paths are Bounded by $G(N)$ 59

5.3. Analysis of G(N) 62

5.4. Experimental Results 65

5.5. Conclusion 67

CHAPTER 6: CONCLUSION 68

6.1. Summary of Results 68

6.2. Open Questions 69

REFERENCES 70

LIST OF TABLES

TABLE 4.1: Critical path length (95% confidence intervals) and ratios of average critical path lengths across methods for different RMAT parameters. (Bolded numbers highlight critical path length ratios greater than 1.15.) 55

TABLE 4.2: Graph basic statistics, critical path length (95% confidence intervals), and ratios of average critical path lengths across methods for several real world graphs. (Bolded numbers highlight critical path length ratios greater than 1.15.) 56

LIST OF FIGURES

FIGURE 2.1: OR Subgraph 10

FIGURE 3.1: Application leading to a 5×4 9-pt stencil graph 15

FIGURE 3.2: Odd Cycle Instance and its Optimal Coloring 19

FIGURE 3.3: Optimal Coloring of 2 Neighboring Cycles 21

FIGURE 3.4: Instance Samples 32

FIGURE 3.5: 2D Results (All Instances) 33

FIGURE 3.6: Performance Profile for 2DS-IVC: *maxcolor* broken down per dataset 34

FIGURE 3.7: 3D Results (All Instances) 36

FIGURE 3.8: Performance Profile on 3DS-IVC: *maxcolor* broken down by dataset 37

FIGURE 3.9: Performance Profiles with ILP 41

FIGURE 3.10: Scatter plot of number of colors and execution time of the STKDE application. Each scatter plot presents different coloring algorithm. A linear regression line shows positive corelation between number of colors and runtime in all 6 cases. 42

FIGURE 4.1: The execution time of dataflow algorithm depends on the random number generation. Black edges highlight the order of the dataflow, while green edges show the critical path. On a lucky draw, the critical path of the algorithm contains only 2 vertices, while an unlucky draw can have 16 vertices in its critical path. 45

FIGURE 4.2: Cumulative Density Function of the longest chain induced by `Uniform`, `Expnential` and `Linear` on RMAT Graph with a = 0.10, b = 0.20, c = 0.50, d = 0.20 for different values of edge factor ef. The different values of edge factor show almost identical patterns for the length of the critical path. 49

FIGURE 4.3: Cumulative Density Function of the longest chain induced by `Uniform`, `Expnential` and `Linear` on RMAT Graph with a = 0.42, b = 0.19, c = 0.19, d = 0.02 for different values of edge factor ef. The different values of edge factor show almost identical patterns for the length of the critical path. 50

FIGURE 4.4: Cumulative Density Functions of longest chain on real-world Graphs. All graphs (except ca-HepTh and roadNet-PA) show a major difference in critical path length across methods: `Exponential` and `Linear` have much shorter critical paths than `Uniform`. 57

FIGURE 5.1: Experimental Results for Long Paths in $G_{N,P}$ 65

CHAPTER 1: INTRODUCTION

Graphs are a fundamental tool in mathematics used to model processes found in almost every domain of science and technology. Graphs are a collection of nodes and edges, in which nodes are connected by edges. Nodes typically represent objects and edges represent relationships between them. Graphs have numerous applications to physical systems, social networks, and computational tasks.

Graphs are used to model transportation, in which cities are represented by nodes and the roads connecting them are edges. They are used in engineering to build and test automobiles and planes. Not only is the design of the vehicle modeled by a collection of neighboring parts but also its relationships with the surrounding environment. Facebook is a well-known social network social network, where people and their friendships are represented as a graph. In biology, how proteins regulate and control one another is expressed in a graph called a protein-protein interaction network.

Developing the graph model of an application is often helpful to gain insight into the problem but is usually not sufficient to solve most real-world problems. The study of graphs is important because additional tools are needed to use these models effectively. We use the maximum flow algorithm to identify potential traffic congestion when designing roadways and other infrastructure. We also use the shortest path algorithm to efficiently get from point A to point B. Computational fluid dynamics is essential to ensure a plane will fly correctly before its built. Protein-protein interaction networks and social networks are both studied for their community structure.

The cost of graph analysis significantly increases with the size and complexity of the model. Complex models require substantial computational resources because many

state-of-the-art algorithms are severely non-linear, so the time it takes for analysis greatly increases on large graphs. Therefore, it has become increasingly important to optimize algorithmic complexity for large graphs because graph analysis has become a staple of modern science and technology. In this paper we will present methods that use graphs to improve parallel computing as well as parallel computing methods to make graph analysis faster.

Parallel computing is the natural approach to improve the performance of algorithms on large graphs. In recent years computers have not become faster sequentially; however, transistors have become smaller, so more cores have been added to CPUs. Dennard scaling is a formal explanation of this phenomena [1]. GPU computing has increased in popularity for this same reason - GPUs are designed to take as many simple tasks as possible and complete them simultaneously. Overall, computing performance has increased precisely because more tasks are completed in parallel.

Task graphs are data structures used to illustrate a sequence of tasks and their dependencies. Each node is a task that needs to be completed, and each directed edge represents what task needs to be completed before the other is started. It is necessary to decide which processor will execute a given task and when the execution of that task will start. The objective of task graph scheduling is often to minimize the total runtime of a task graph on a given platform [2, 3].

Resource allocation attempts to minimize the average runtime of a given task using the configuration of the machine. The system allocates the workload based on how many resources are currently being used and how costly it is to communicate the output of a task [4]. For example, when mass producing a car the question of resource allocation would asks us, "What is the best way to setup my assembly line so that cars are produced as quickly as possible?" These are formally known as the flow shop problem [5] and open shop scheduling problem [6]. The fundamental difference between resource allocation and task graph scheduling is that resource allocation

uses properties of the machines for optimization, whereas task graph scheduling uses properties of the tasks.

Conflict graphs model tasks that cannot be completed at the same time [7]. This model closely resembles the problem of classical graph coloring. The classical problem of graph coloring assigns color to each vertex so that no adjacent vertices share the same color, and the objective is to minimize the number of colors used. Coloring a graph with intervals assigns an interval of consecutive colors to each vertex and has the added restriction that no adjacent vertices can share any color in their respective intervals.

The problem of scheduling tasks is fundamentally the same as coloring a graph with intervals. Each task can be represented as a vertex and task precedence can be represented as a directed edge. We can refine this model by adding a weight to each vertex. This weight is proportional to the computational requirement for each task, which determines how large an interval must be when coloring the graph. The scheduling objective of minimizing runtime becomes minimizing the total number of colors in the graph. Because all intervals must be consecutive, this objective is similar to minimizing the largest color assigned to each vertex. Time in scheduling is analogous to consecutive colors in interval coloring.

Executing graph algorithms in a parallel or distributed context is a challenging problem because of race conditions. Race conditions are the result of a machine executing tasks simultaneously that needed to be processed sequentially. And often in graph algorithms, two neighbors can not be processed simultaneously. Solving race conditions is expensive so many algorithms ignore them by checking the solution after it has already been given. Alternatively, dataflow algorithms solve this problem by using a partial order on the graph. Dataflow algorithms in this context can be posed as a distributed interval coloring problem.

Classical graph coloring has an established history of theorems and problem vari-

ants, but most state-of-the-art algorithms are computationally expensive, and many depend on exploiting the structure of a particular type of graph [8]. Although interval coloring does not appear to have much existing literature, we believe that it is a valuable tool to approach task graph scheduling and resource allocation and may have other applications outside of parallel computing. Because interval coloring is more complex than traditional graph coloring, it is very likely that this problem is simply understudied.

1.1 Problem Statement

We are interested in the problem of interval coloring because it is relevant to scheduling parallel applications. In this paper we explore the problem of interval coloring from three different perspectives. First, we look at a particular case of interval coloring for stencil graphs. Next, we search for good interval colorings for distributed graphs. Lastly, we seek to generalize this result to random graph models.

Stencil graphs appear naturally in the parallelization of applications where the location of an object in a space affects the state of neighboring objects. Rectilinear decompositions of a space generate conflict graphs that are 9-pt stencils for 2D problems and 27-pt Stencils for 3D problems. We show that the 5-pt stencil and 7-pt stencil relaxations of the problem can be solved in polynomial time.

We prove that the decision problem on 27-pt stencil is NP-Complete. We discussed approximation algorithms with a ratio of 2 for the 9-pt stencil case, and 4 for the 27-pt stencil case. We identify two lower bounds for the problem that are used to design heuristics. We evaluate the effectiveness of several different algorithms experimentally on a set of real instances. Furthermore, these algorithms are integrated into a real application to demonstrate the soundness of the approach. This work on interval coloring stencil graphs was published in the 36th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2022) [9].

We also investigated how a distributed dataflow graph algorithm obtains a partial

order and how one could favor orders with shorter long chains. Most dataflow algorithms obtain their order by having each vertex of the graph pick a uniformly random number in [0, 1) and order the vertices based on that number. We believe that this type of order could lead to long chains in graphs with dense regions such as small world graph.

We design two alternative ways of generating an order similar to largest degree first. We study the behavior of these different algorithms on a wide range of randomly generated RMAT graphs and on a set of real-world graphs. We show in simulation that our ordering methods can significantly reduce the length of the longest chain. This work on dataflow algorithms was published in the 13th IEEE Workshop Parallel / Distributed Combinatorics and Optimization (PDCO 2023) [10].

Furthermore, we search for a precise explanation of the performance difference found in both the RMAT and real-world experiments. We provide a formal proof for $G_{N,P}$ graphs as a preliminary.

1.2 Document Structure

In Chapter 2, we provide essential tools to better understand the work that has been completed. In Section 2.1, we review the concept of a valid k-coloring and the definition of chromatic number. We characterize several special types of graphs and reprove some elementary results associated with classical coloring. We establish basic proof techniques using NP-Completeness as well as motivate the use of heuristics. In Section 2.2, we develop the concept of coloring with intervals as a natural extension of classical graph coloring. We review existing literature that better illustrates the similar nature of scheduling, edge orientation, and coloring problems. We also motivate developing better partial orders for dataflow algorithms.

In Chapter 3, we study the formal problem of coloring with interval graphs which are 9-pt 2D stencils and 27-pt 3D stencils. We formally define the combinatorial problem in Section 3.2. We study special cases in Section 3.3 where we show how to

color important particular graphs such as cliques, bipartite graphs, and odd cycles. This analysis gives us lower bounds useful to analyse the stencil problem. We prove in Section 3.4 that the problem of interval coloring of 27-pt 3D stencil with a small number of colors is NP-Complete. Section 3.5 provides various greedy heuristics based on the analysis of the problem. It also provides an approximation algorithm for the problem with a ratio of 2 for the 9-pt stencil problem and of 4 for the 27-pt stencil problem; and greedy post optimizations. All the methods are evaluated on some instances from spatio-temporal analysis in Section 3.6. In Section 3.7, we integrate our heuristics in a Space-Time Kernel Density Estimation application [11] and show that the number of colors derived by the heuristics correlates with the runtime of the application.

In Chapter 4, we present our study of dataflow algorithms. Section 4.2 explains precisely how dataflow algorithms work and provides a model of the problem as a graph coloring problem. Section 4.3 presents new ways to generate orderings for dataflow algorithms and gives a theoretical argument for why they are sound. Section 4.4 studies the behavior of the algorithms on random RMAT graphs and shows that our methods perform usually better. Section 4.5 studies the behavior of these algorithms on real-world graphs and shows that our methods perform usually better.

In Chapter 5, we study the probability of long paths in random graphs. We provide a proof that as long as $P = \frac{c_1 \log N}{N}$ the probability of a long path in $G_{N,P}$ graphs goes to 0 when N is sufficiently large. We also provide experimental results that reinforce this claim in Section 5.4.

Chapter 6 contains some concluding remarks to the dissertation. We provide an overview of the work that was completed and any open questions that remain.

CHAPTER 2: PRELIMINARIES

2.1 Basic Definitions and Theorems

Definition 2.1.1. *Let $G = \{V, E\}$ be a graph, such that V is the set of vertices and E is the set of edges. We often use v to represent a vertex in V and (u, v) as an edge in E.*

Definition 2.1.2. *A vertex v is said to be adjacent to a vertex u if $(u, v) \in E$.*

Definition 2.1.3. *A graph is undirected if every pairwise edge from u to v is the same edge from v to u. A graph that is not undirected is said to be directed.*

The notation of an edge (u, v) is intended to be an unordered tuple in the context of undirected graphs and an ordered tuple in the context of directed graphs.

Definition 2.1.4. *A graph is said to have a proper (or valid) k-coloring if there exists a set of colors $X = \{c_1, c_2, ..., c_k\}$ so that $\{X_{c_1}, X_{c_2}, ..., X_{c_k}\}$ is a partition of V with the property that for every $c_i \in X$ if u and v are distinct vertices of X_{c_i}, then $(u, v) \notin E$.*

Note: $\{X_{c_1}, X_{c_2}, ..., X_{c_k}\}$ is a partition of V if $\bigcup_{i=1}^{k} X_{c_i} = V$, and for every $v \in V$ if $v \in X_{c_i}$, then $v \notin X_{c_j}$ for all $j \neq i$.

Definition 2.1.5. *The chromatic number of G denoted $\chi(G)$ is the smallest value of k for which G has a valid k-coloring.*

Definition 2.1.6. *$U \subset V$ is independent if for every $u \in U$, u is not adjacent to any other vertex in U.*

Definition 2.1.7. *We say that a graph is bipartite if V can be partitioned into 2 disjoint and independent subsets $\{U, W\}$, so that every vertex $u \in U$ is adjacent to at least one vertex $w \in W$.*

Theorem 2.1.8. *Bipartite graphs can be colored using only 2 colors.*

Proof. The sets provided by the definition of bipartite $\{U, W\}$ can be used as color classes. These sets are disjoint by definition, so this coloring valid. Since there are only 2 classes, the graph is 2-colorable. □

Definition 2.1.9. *A path is a sequence of edges required to traverse from one vertex to another using only connected vertices.*

Definition 2.1.10. *We call a graph a cycle if there exists a path from any given vertex back to itself without repeating other any vertices. The number of edges in this path is the length of the cycle.*

Theorem 2.1.11. *Odd cycles are* 3*-colorable.*

Proof. Let G be a cycle of length k, such that k is odd. Clearly, $k = 1$ and $k = 3$ are 3-colorable, so we can assume $k \geq 5$. We will now show that a cycle of length $k + 2$ is 3-colorable by induction on k.

Now, suppose G is an odd cycle of length $k + 2$. Since $k \geq 5$, we can take vertices u and v, so they do not not share both of the same neighbors. Consider $G \setminus \{u, v\}$ by collapsing each of their respective edges. Remove u and the edges containing u, and add an edge so that the neighbors of u will now be adjacent. Repeat this for v. $G \setminus \{u, v\}$ results in a cycle with 2 less edges. Since $G \setminus \{u, v\}$ is an odd cycle of length k, $G \setminus \{u, v\}$ is 3-colorable by the induction hypothesis.

Let X is a valid 3-coloring for a cycle of length k. Remove an edge (p, q) anywhere on $G \setminus \{u, v\}$ and insert u between p and q, so that both p and q are adjacent to u. Color u differently from both p and q. This is a valid 3-coloring. Repeat this process when adding back v. The resulting graph is a cycle of length $k + 2$ with a valid 3-coloring. Therefore, cycles of odd length are 3-colorable by induction. □

Definition 2.1.12. $U \subset V$ *is clique if for every* $u \in U$*,* u *is adjacent to every other vertex in* U*.*

Theorem 2.1.13. *A clique of size k requires k colors.*

Proof. Let G be a clique of size k. Suppose for the sake of contradiction X is a valid $k-1$ coloring of G. By the Pigeonhole Principle, there exist 2 vertices with the same color, say u and v. However, u and v are adjacent by definition of a clique. This is a contradiction; therefore, X is not a valid $k-1$ coloring of G. □

Definition 2.1.14. *The neighborhood of a vertex, $n(v)$, is the set of all other vertices adjacent to v. If $u \in n(v)$, then u is called a neighbor of v.*

Definition 2.1.15. *The degree of a vertex, $\delta(v)$, is the number of neighbors of v. The largest degree of any vertex in the graph is notated $\Delta(G)$.*

Theorem 2.1.16. *Graphs can be colored in $\Delta(G) + 1$ colors.*

Proof. Let G be a graph. We will prove G can be colored in $\Delta(G) + 1$ colors by constructing a greedy algorithm. Let v be an uncolored vertex. Since $d(v) < \Delta(G)+1$, there are at most $\Delta(G)$ used colors in the neighborhood of v. Therefore, there is a color that has not already been used by any neighbor of v. Use that color for v. Repeat this process until all vertices are colored. □

Theorem 2.1.17. 3*-coloring is NP-Complete by reduction to* 3*-SAT.*

Proof. The proof is a Karp reduction that follows directly from Lemmas 2.1.18 and 2.1.19. □

Lemma 2.1.18. 3*-coloring* $\in$ *NP.*

Proof. Suppose we have a 3-coloring of a graph. For each edge (u, v) check if the color of u is different from the color of v. Since we are checking at most E edges, the validity of the coloring can be determined in complexity $O(E)$. □

Lemma 2.1.19. 3*-SAT* $\propto$ 3*-coloring.*

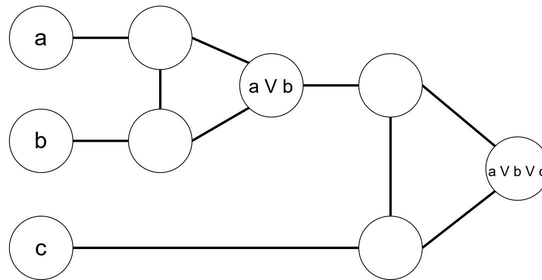


Figure 2.1: OR Subgraph

Proof. Let $\{x_1, x_2, ..., x_n\}$ be variables and $\{C_1, C_2, ..., C_m\}$ be clauses of a $3-$SAT formula ϕ. Each clause is a 3-tuple of variables, such that $C_j = (a \vee b \vee c)$ is True if either a or b or c is True. Conversely, $C_j = (a \vee b \vee c)$ is False if neither a nor b nor c is True. We will construct G, so that G is 3-colorable if and only if ϕ is 3-satisfiable.

Let $\{T, F, B\}$ be a clique of size 3. For each variable x_i, add vertices v_i and $\overline{v_i}$ to the graph with the following edges: (B, v_i), $(B, \overline{v_i})$, and $(v_i, \overline{v_i})$. If G is 3-colored either v_i or $\overline{v_i}$ is colored the same as T. If v_i is colored the same as T, then x_i is interpreted as True. If $\overline{v_i}$ is colored the same as T, then x_i is interpreted as False.

For each $C_j = (a \vee b \vee c)$, construct a subgraph that has the following property: the output node of the subgraph is colored the same as T when C_j is True and the output node is colored the same as F when C_j is False. Figure 2.1 depicts a subgraph with this property. If we connect the node that represents the output $(a \vee b \vee c)$ to both F and B. We get our desired construction.

We will now show that ϕ is 3-satisfiable implies G is 3-colorable. If x_i is assigned True, then color v_i the same color as T and $\overline{v_i}$ the same color as F. For each clause our subgraph can be colored so that the output node is True.

We will now prove the converse that G is 3-colorable implies ϕ is 3-satisfiable. If v_i shares the same color as T, let x_i be True. Let $C_j = (a \vee b \vee c)$ be a clause. Suppose a, b, and c are all False to reach a contradiction. Hence, the output node of the corresponding subgraph must share the same color as F. However, the output node

is adjacent to both B and F. Since G is 3-colorable, the output must share the same color as T. This is a contradiction, so either a or b or c must be True, which implies C_j is True.

Therefore, G is 3-colorable if and only if ϕ is 3-satisfiable.

□

2.2 Related Works

The problem of interval coloring has a long-established history with multiple variants and classical applications. Bandwidth problems [12, 13], scheduling problems [14], and timetabling problems [15] are just a few applications.

Since the complexity of the interval coloring problem for general graphs is known to be NP-Hard [16], several authors have provided bounds on the generalized chromatic number [17]. However, these bounds are not useful in practical applications; therefore, polynomial algorithms for special classes of graphs become desirable. Bipartite graphs, complete graphs, chordal graphs, interval graphs, stars, and trees have already been investigated. However, 9-pt 2D stencil graph and 27-pt 3D stencil were previously unexplored, as far as the authors know.

Greedy algorithms are a staple of heuristics to provide solutions to graph coloring problems since graph coloring is NP-Complete [18]. For classic graph coloring problems greedy algorithms pick vertices of the graph in an arbitrary order and allocate the lowest color that does not conflict with the neighbors that have already been colored. A classic guarantee of greedy coloring is that they use at most $\Delta + 1$ colors where Δ is the maximum degree in the graph.

Some classic greedy algorithms use a particular ordering of the vertices which hopefully provide better colorings than arbitrary orders [19]. Popular orderings are Largest First [20], and Smallest Last [21]. Some post optimization techniques have proven to be particularly effective, such as recoloring [22].

Scheduling, edge orientation, and coloring problems are fundamentally related. In

a typical task graph scheduling problem [23], the order of tasks is known in advance and the problem is to decide when and where the tasks will run given a list of dependency constraints. An interesting result is that list scheduling [24] always guarantees to get an application executed on P processors quicker than $\frac{\sum_{v\in V} w(v)}{P} + max_{c\in allchains(G)} \sum_{v\in c} w(v)$. The second term is the length of the longest chain in the graph, which is optimized by minimizing the number of colors.

Another problem is the edge orientation problem. The Gallai-Hasse-Roy-Vitaver theorem proved that the maximum path length in an oriented graph is always greater than one plus the chromatic number of its unoriented counterpart (and equal for the optimal orientation) [25]. On weighted graphs, most edge-orientation problems attempt to minimize maximum weighted outdegree [26] as opposed to maximum path length of an acyclic orientation. There are distributed algorithms to minimize the number of colors for graphs with particular structures. For instance, if the edges can be oriented so that every vertex has an outdegree less than or equal to 1, then the Cole-Vishkin algorithm can be applied [27].

Even though the classic problem of coloring graph is polynomial for particular categories for graphs [28], it is NP-Complete for arbitrary graphs [16]. On arbitrary graphs, the problem is even not polynomially approximable [29] and often the best guarantee that can be made is that greedy algorithms can always achieve a coloring using fewer than $\Delta + 1$ colors [19]. In practice, it has been reported that selecting orderings of vertices can generate better colorings [21, 30, 20]. We leverage that intuition in this particular work where we generate orderings using some graph property to reduce the total number of colors.

Coloring graphs with intervals has received little consideration in the past. The general NP-Completeness result holds on arbitrary graphs and the problem is often studied from a radio spectrum allocation perspective [31]. Recently, we studied the problem of coloring stencil graphs with interval and provided NP-completeness and

approximation results [9].

The complexity of dataflow algorithms for maximal independent set and matching was shown to be polylogarithmic with high probability when processing random graphs on PRAM machines [32]. However, they did not attempt to reduce the critical path length by adjusting the randomized algorithm.

CHAPTER 3: INTERVAL COLORING WITH STENCILS

3.1 Introduction

In parallel computing, a central question is to decide when each task should be run. There are two fundamental models to reason with this problem. The first is the Parallel Task Graph model which encodes what the tasks are and the precedence dependences between tasks. Though, in some applications, the order in which the tasks are run can be changed, as long as some sets of tasks do not run concurrently. To model this type of application, the tasks and their conflicts are represented as an undirected graph in which vertices are tasks, and edges represent the non-concurrency between two tasks. The classic optimization problem to make the execution more efficient is a graph coloring problem, which is NP-Hard in the general case [16].

In the classic graph coloring problem, each vertex of the graph needs to be allocated one color (an integer) so that each pair of neighboring vertices have different colors. This model is appropriate when one does not have a good idea of the runtime for each individual task, which happens frequently. Though in some applications we have a precise idea of how much work is required by a particular task. In these cases, the problem of scheduling the tasks is better modeled by giving each task not a single color, but an interval of colors with length proportional to the length of the task. This problem is to color the vertices of a graph with intervals. In the general case, this problem is harder than the classic graph coloring problem and is also NP-Hard even though it provides a more accurate model.

While the problem is NP-Hard on general graphs, certain types of applications are only concerned about particular categories of graph. In this chapter, we study the problem of coloring with intervals the vertices of stencil graphs. In particular, we are

interested in 9-pt 2D stencils and in 27-pt 3D stencils.

These problems appear in applications where objects are located in space and can impact the state of nearby objects. Imagine an application in 2D space where the impact of objects within a given radius follow the behavior of complex equations. When making this application parallel, one may want to partition the space and have each region of the space be a particular task. See Figure 3.1 for reference. The figure depicts a grid of 5×4 tasks. The blue object will impact the three objects within the radius of the blue circle. So when processing the region that contain the blue object, one can not process any other region that may impact the same objects. If the partition of the region is made to be rectilinear [33] and no partition is smaller than twice the radius of impact, then a region can not be processed at the same time as any of its 8 neighbors. The underlying graph of conflict is a 9-pt 2D stencil. The nodes can be weighted with an estimation of the processing time of the region. In the figure, the nodes are weighted by the number of objects in the region. This type of structure can appear in various scientific codes, including n-body solvers, bird flocking simulations [34], or visualization of spatio-temporal data [11].

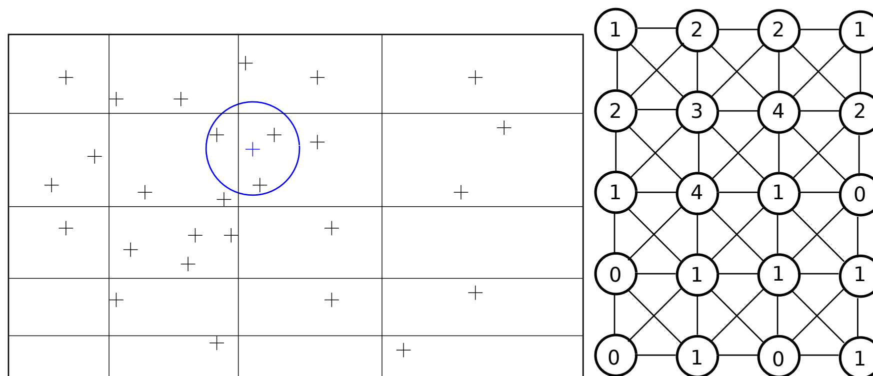


Figure 3.1: Application leading to a 5×4 9-pt stencil graph

3.2 Interval Coloring Problem of Stencils

3.2.1 Problem Definition

We define first the general problem of graph coloring vertices with intervals.

Definition 3.2.1 (Interval Vertex Coloring (IVC))**.** *Let $G = (V, E)$ be an undirected graph and $w : V \to Z^+$ be a weight function that associates vertices of the graph to positive (or null) weights.*

An interval coloring of the vertices of G is a function $start : V \to Z^+$. We say that vertex v is colored with the open interval $[start(v), start(v) + w(v))$. For the coloring to be valid, neighboring vertices must have disjoint color intervals $\forall (a, b) \in E, [start(a), start(a) + w(a)) \cap [start(b), start(b) + w(b)) = \emptyset$. A particular coloring start of vertices is said to use $maxcolor = \max_{v \in V} start(v) + w(v)$ colors.

The optimization problem is to find a coloring start that minimizes $maxcolor$. We will denote the optimal value of $maxcolor$ as $maxcolor^*$.

We will slightly abuse the w notation to extend it to sets of vertices: for instance, $w(x, y, z) = w(x) + w(y) + w(z)$.

We are particularly interested in restrictions of the problems where the graph is a 9-pt 2D stencil or a 27-pt 3D stencil.

Definition 3.2.2 (2DS-IVC)**.** *An IVC problem where graph G is a 9-pt 2D stencil, that is to say it is composed of $X \times Y$ vertices laid on a 2D grid such that two vertices (i, j) and (i', j') are connected by an edge if and only if $|i - i'| \leq 1$ and $|j - j'| \leq 1$.*

Definition 3.2.3 (3DS-IVC)**.** *An IVC problem where graph G is a 27-pt 3D stencil, that is to say it is composed of $X \times Y \times Z$ vertices laid on a 3D grid such that two vertices (i, j, k) and (i', j', k') are connected by an edge if and only if $|i - i'| \leq 1$ and $|j - j'| \leq 1$ and $|k - k'| \leq 1$.*

Without loss of generality, we will assume that $X > 1$, $Y > 1$, and $Z > 1$ for both 2DS-IVC and 3DS-IVC instances. If one of the dimensions was equal to 1 in 3DS-IVC, the instance can be thought as an instance of 2DS-IVC. And if one of the dimension was equal to 1 in 2DS-IVC, the graph would be a chain which, as we will see, is a polynomial case.

In general, vertices are indexed from 1; so the first task of 2DS-IVC is $(1,1)$ and the last task is (X,Y).

3.3 Special Case Analysis

Since we are in particular interested in solving the 2DS-IVC and 3DS-IVC problems, it is important to analyze graphs structures that can be embedded in a 9-pt or a 27-pt stencil. Indeed, for any instance of IVC (and therefore of 2DS-IVC or 3DS-IVC), the optimal coloring of any subgraph contained in the graph G (obtained for instance by removing vertices or edges from G) is a lower bound of the optimal number of color of G.

3.3.1 Cliques

Cliques are some of the easiest graphs to color. Because all vertices are connected to all the other vertices, no vertices can share any color with any other vertices in the graph. Therefore, if $G = K_n$ is a clique of size n, it is optimal to color the graph with $maxcolor^* = \sum_{v \in V} w(v)$ colors. One can easily build such a coloring by listing vertices in any order and greedily allocating the color inteveral with the lowest available $start(v)$; with a complexity of $\Theta(V)$.

Cliques are particularly important for our stencil problems because 2DS-IVC contains many K_4 and 3DS-IVC contains many K_8. So the sum of weight for each block of 4 neighboring vertices is a lower bound of 2DS-IVC ($\forall 0 \leq i < X, 0 \leq j < Y, maxcolor^* \geq w(i,j) + w(i,j+1) + w(i+1,j) + w(i+1,j+1)$) and the sum of weight of each block of 8 neighboring vertices is a lower bound of 3DS-IVC ($\forall 0 \leq i < X, 0 \leq j < Y, 0 \leq k < Z, maxcolor^* \geq w(i,j,k) + w(i,j+1,k) + w(i+1,j,k) + w(i+1,j+1,k) + w(i,j,k+1) + w(i,j+1,k+1) + w(i+1,j,k+1) + w(i+1,j+1,k+1)$).

3.3.2 Bipartite Graph

If the graph G is bipartite, that is to say if vertices can be partitioned in two sets A and B such that all edges have one extremity in A and one extremity in B, then

the graph is easy to color with intervals. Each edge in the graph provides a trivial lower bound for the number of colors $maxcolor^* \geq w(i) + w(j), \forall (i,j) \in E$.

A simple algorithm achieves a coloring with $maxcolor^* = \max_{(i,j)\in E} w(i) + w(j)$. If $i \in A$, color it with $start(i) = 0$ in the interval $[0, w(i))$. If $j \in B$, color it with $start(j) = maxcolor^* - w(j)$ in the interval $[maxcolor^* - w(j), maxcolor^*)$. This algorithm is correct because all edges are between a vertex of A and a vertex of B: the color interval of the vertices are disjoint by definition of $maxcolor^*$.

The algorithm requires two linear passes over the graph: one to identify A and B and compute $maxcolor^*$; and one to set the colors of all vertices. The algorithm has a complexity of $\Theta(E)$.

Bipartite graphs are quite important to 2DS-IVC and 3DS-IVC because each 9-pt stencil contains a 5-pt stencil which is bipartite. Similarly, each 27-pt stencil contains a 7-pt stencil which is also bipartite. We will see that this property enables us to build approximation algorithms for these problems. Also any chain and even cycles embedded in the stencil is bipartite.

3.3.3 Odd Cycles

Graphs that are not bipartite contain at least one cycle of odd length. It turns out that odd cycles can have optimal interval colorings that are strictly greater than the largest weight of the any clique in the graph. Consider the odd cycle embedded in a 2D stencil presented in Figure 3.2, the clique of largest weight is 25, but the optimal coloring is 30. As such, understanding how to color odd cycles with intervals will yield new lower bounds on optimal interval coloring of any graph, including 9-pt stencils and 27-pt stencils.

Because in this case G a cycle, the neighbors of vertex x are denoted as $x - 1$ and $x + 1$; in other words, indices are understood modulo $|V|$.

Definition 3.3.1 (maxpair)**.** *Let maxpair be the maximum sum of any 2 consecutive terms:* $maxpair = \max_i w(i, i+1)$

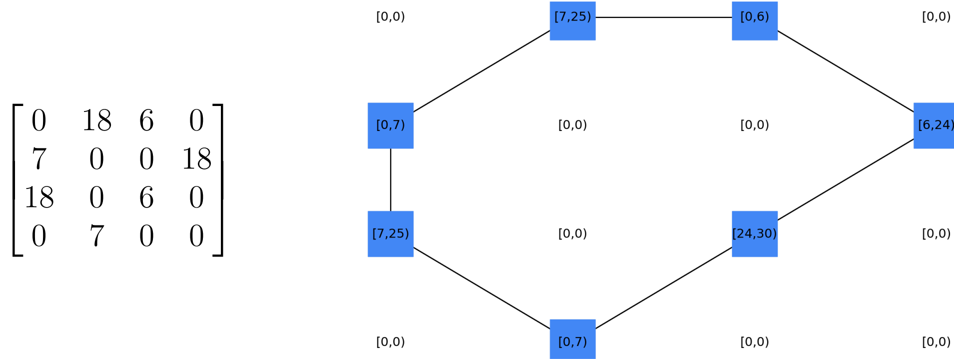


Figure 3.2: Odd Cycle Instance and its Optimal Coloring

Definition 3.3.2 (minchain3)**.** *Let* $minchain3$ *be the minimum sum of any 3 consecutive terms:* $minchain3 = \min_i w(i, i+1, i+2)$

Theorem 3.3.3. *If* G *is an odd cycle, we have* $maxcolor^* = \max(maxpair, minchain3)$

We prove this theorem by proving that this value of $maxcolor^*$ is feasible and is also a lower bound on the number of colors in two seperate lemmas.

Lemma 3.3.4. *If* G *is an odd cycle, there is an algorithm that yields* $\max(maxpair, minchain3)$ *colors. In other words* $maxcolor^* \leq \max(maxpair, minchain3)$

Proof. Without loss of generality, the three consecutive vertices that give $minchain3$ are assumed to be 0, 1, and 2.

We color vertex 0 with $start(0) = 0$ (and therefore with interval $[0, w(0))$); we color vertex 1 with $start(1) = w(0)$ (and therefore with interval $[w(0), w(0,1))$); and we color vertex 2 with $start(2) = \max(maxpair, minchain3) - w(2)$ (and therefore with interval $[\max(maxpair, minchain3) - w(2), \max(maxpair, minchain3))$).

For the remaining vertices x, if x is odd, we color it with $start(x) = 0$ (and therefore with interval $[0, w(x))$); if x is even, we color it with $start(x) = \max(maxpair, minchain3) - w(x)$ (and therefore with interval $[\max(maxpair, minchain3) - w(x), \max(maxpair, minchain3))$).

Obviously, this coloring uses exactly $\max(maxpair, minchain3)$ but we need to argue that it is correct. By construction, vertices 0, 1, and 2 do not have intersecting color intervals.

For all vertex $x > 1$, the color intervals of x and $x+1$ do not intersect because one of the interval starts on 0 and the other ends on $\max(maxpair, minchain3)$ and the length of $[0, \max(maxpair, minchain3))$ is larger than $w(x, x+1)$ by construction. $\square$

Lemma 3.3.5. *If G is an odd cycle, $maxcolor^* \geq \max(maxpair, minchain3)$*

Proof. We can assume $maxpair < minchain3$. (If $minchain3 \leq maxpair$, then the lemma is obviously true since $maxpair$ is a lower bound of number of colors on any graph.) Let $K = minchain3$. The proof is by contradiction: Suppose for that G can be colored in $K-1$ colors and assume we have a valid coloring $start$. Let $i(x) = [start(x), start(x + w(x))$.

We will once again assume without loss of generality that $w(0, 1, 2) = minchain3$. So we have $w(0, 1, 2) \leq w(x, x+1, x+2)$ for all $x \in V$.

We have $i(0) \cap i(2) \neq \emptyset$ because the Pidgeonhole Principle: there are only $w(0, 1, 2) - 1 = K - 1$ colors available; and, because i is valid, we have $i(0) \cap i(1) = \emptyset$ and $i(2) \cap i(1) = \emptyset$.

Since $i(0)$ and $i(2)$ intersect, but do not intersect with $i(1)$, $i(0)$ and $i(2)$ must be on the same side of $i(1)$. Without loss of generality, we can assume that $i(1)$ is before $i(0)$ and $i(2)$. If it is not true, we can transform the coloring so that color c becomes color $k - 1 - c$. And since 1 is only neighbor with 0 and 2, we can assume that $i(1) = [0, w(1))$. We say that 1's coloring is 0-aligned.

$w(3) \geq w(0)$ because $w(1, 2, 3) \geq w(0, 1, 2)$ since $(0, 1, 2)$ is the minimum chain of length 3. Hence, $i(3) \cap i(1) \neq \emptyset$ since $i(3) \cap i(2) = \emptyset$ and $i(1) \cap i(2) = \emptyset$. Therefore, $i(1)$ and $i(3)$ are on the same side of $i(2)$ since $i(1)$ is 0-aligned, we can assume WLOG that $i(2) = [K - 1 - w(2), K - 1)$. we say that 2's coloring is $K - 1$-aligned.

This argument is true for any chain of three vertices: $\forall x, i(x) \cap i(x+2) \neq \emptyset$. The same argument holds by induction. For all odd x, we have $i(x) = [0, w(x))$. And for all even x we have $i(x) = [K-1-w(x), K-1)$. We have $i(n-1) = [K-1-w(n-1), K-1)$ because $n - 1$ is even. The Pidgeonhole Principle implies that $n - 1, 0, 1$ has their

interval intersect. But since $i(n-1)$ and 1 do not intersect, and $i(0)$ and $i(1)$, then $i(n-1)$ and $i(0)$ must intersect. Hence, the solution is not valid. $\square$

Odd cycles provide a new lower bound on the optimal coloring of the 2DS-IVC and 3DS-IVC: the maximum $minchain3$ of any odd cycle embedded in the stencil. However, it does not appear to be easy to identify the odd cycle of maximum $minchain3$ in an instance of 2DS-IVC. There are an exponential number of odd cycles; so one would need something of lower complexity than simply listing them.

3.3.4 Lower bounds are not tight

We now have two separate lower bounds applicable to our stencil graphs. Cliques provide one lower bound and odd cycles provide the other one. We exhibit now (in Figure 3.3) an instance whose optimal coloring uses stricly more color than either lower bounds.

Figure 3.3: Optimal Coloring of 2 Neighboring Cycles

The instance features two odd cycles that have two of their respective vertices

neighbor each other. The maximum clique is 14 while the $minchain3$ of either of the cycle is 14. Yet, the optimal coloring is 17. (We confirmed the optimal coloring with an integer linear program.)

3.4 NP-Completeness

We will prove in this section that the decision version of the 3DS-IVC problem is NP-Complete. The core of the proof is to show that the problem is harder than Not-All-Equal 3-SAT.

An instance of Not-All-Equal 3-SAT (NAE-3SAT) is qualified by n binary variables used in m groups of 3 variables. The instance is positive if there is an assignment of true or false to each variable so that in each of the m groups at least one variable is true and at least one is false. This variant of 3SAT is known to be NP-Complete [35]. NAE-3SAT has two of properties which makes it easier to use in many reductions: 1) there is no need for negation of a variable in the instance of NAE-3SAT like we have in 3SAT; and 2) if an assignment solves the instance, then the negation of that assignment also solves the instance.

Lemma 3.4.1. *3DS-IVC* $\in$ *NP*

Proof. A solution for 3DS-IVC is an interval of colors for each vertex. This can be encoded as 2 integers, and they are easily bounded between 0 and $\sum_{i=0}^{n} w(i)$, where $w(i)$ is the weight of the vertex i in 3DS-IVC. This sum can be encoded in a polynomial number of bits. This is a trivial bound, but it does show the solution is in polynomial space.

Given a solution for 3DS-IVC we can check to see if it is correct in polynomial time. We just need to verify that no adjacent edges have overlapping scheduled intervals. More precisely, we are checking, $\forall (u,v) \in E, [start(u), start(u) + w(u)) \cap [start(v), start(v) + w(v)) = \emptyset$. Since $|E| \leq \frac{n(n-1)}{2}$ is polynomial for arbitrary graphs. Checking if two intervals intersect is in $O(1)$. Hence, any solution for 3DS-IVC can

be verified in $O(n^2)$.

Therefore, 3DS-IVC $\in$ NP. $\square$

Lemma 3.4.2. *NAE-3SAT* $\propto$ *3DS-IVC*

Proof. **Constructing an instance 3DS-IVC from an instance of NAE-3SAT in polynomial time.** Let $v_1, v_2, ..., v_n$ be variables that appear in the m clauses of the NAE-3SAT problem, so that for each clause $u_j = (v_{j_1}, v_{j_2}, v_{j_3}), 1 \leq j \leq m$ at least one variable is true and at least one variable is false. Without loss of generality, assume the variables are ordered within the clauses $1 \leq j_1 < j_2 < j_3 \leq n$.

We construct now the corresponding instance of the 3DS-IVC problem to color with $maxcolor = 14$ colors.

We generate a 3D cube of width $2n + 10$, height 9, and depth $2m$. We use (x, y, z) to denote our coordinate system in ^{3}. The weight of each vertex in the 3D cube is either a $0, 3,$ or 7. In other words, $\forall(x, y, z), w(x, y, z) \in \{0, 3, 7\}$. Any value not specified in our construction is set to 0.

We call the following construction a *tube* generated by variable v_i: $\forall(x \leq n, z \leq 2m)$,

$$w(2i-1, 1, z) = \left\{ \begin{array}{ll} 0, & \text{if } z \equiv 1 \pmod 2 \\ 7, & \text{if } z \equiv 0 \pmod 2 \end{array} \right\}$$

$$w(2i-1, 2, z) = \left\{ \begin{array}{ll} 7, & \text{if } z \equiv 1 \pmod 2 \\ 0, & \text{if } z \equiv 0 \pmod 2 \end{array} \right\}$$

We call layer $2j + 1$ "the layer of clause j". For each layer of clause j, we construct the *wire* generated by variable x_{j_1}.

$$w(2j_1 - 1, y, 2j + 1) = 7(\forall y, 2 \leq y \leq 7)$$

$$w(x, 8, 2j + 1) = 7(\forall x, j_1 + 1 \leq x \leq 2n + 1)$$

Similarly, we construct the *wire* generated by variable x_{j_2}.

$$w(2j_2 - 1, y, 2j + 1) = 7(\forall y, 2 \leq y \leq 5)$$

$$w(x, 6, 2j + 1) = 7(\forall x, j_2 + 1 \leq x \leq 2n + 1)$$

Lastly, we construct the *wire* generated by variable x_{j_2}.

$$w(2j_3 - 1, y, 2j + 1) = 7(\forall y, 2 \leq y \leq 3)$$

$$w(x, 4, 2j + 1) = 7(\forall x, j_3 + 1 \leq x \leq 2n + 1)$$

Furthermore, in each odd layer, we explicity describe right hand side of the xy-plane (that is to say for $2n + 1 \leq x \leq 2n + 10$, for $1 \leq y \leq 9$, and for $z = 2j + 1$):

$$W_{2j+1} = \begin{bmatrix} 0 & 7 & 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7 & 0 & 0 & 7 & 0 & 0 & 0 & 7 & 7 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 & 3 & 0 & 0 & 7 \\ 7 & 7 & 0 & 0 & 7 & 3 & 3 & 0 & 0 & 7 \\ 0 & 0 & 7 & 0 & 0 & 0 & 0 & 7 & 0 & 7 \\ 7 & 0 & 0 & 7 & 0 & 0 & 7 & 0 & 0 & 7 \\ 0 & 7 & 0 & 0 & 7 & 7 & 0 & 0 & 0 & 7 \\ 0 & 0 & 7 & 0 & 0 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 7 & 7 & 7 & 7 & 7 & 0 & 0 \end{bmatrix} \tag{3.1}$$

Several desirable properties come from the careful construction of these tubes, wires, and clauses.

The wires connect the tubes to the appropriate "3s" on the right hand side of the clause's layers. All wires have the same parity of length. Meaning, for every variable, the path from the variable to the terminating 3 is congruent to $0 \mod 2$. All wires

have even length in our construction.

Because we are trying to solve the decision problem with $maxcolor = 14$ and each 7 is connected to another 7, each 7 must be scheduled from either $[0, 7)$ or $[7, 14)$. In a chain of $7s$, every other 7 must be scheduled to the same $[0, 7)$ or $[7, 14)$ because adjacent $7s$ cannot overlap in scheduled intervals. In other words, all even $7s$ in a chain must share the same "polarity" by construction. We call the color of $(2i-1, 2, 1)$ the polarity of variable v_i. (If v_i is true, $(2i - 1, 2, 1)$ is colored with interval $[0, 7)$, and the 7s in the tube and wires of v_i have positive polarity. If v_i is false, $(2i-1, 2, 1)$ is colored with interval $[7, 14)$, and the 7s in the tube and wires of v_i have negative polarity.)

In the triangle of $3s$ from the W_{2j+1}, the $7s$ connected to the $3s$ cannot all share the same polarity and be colorable in 14. Suppose without loss of generality that all of the $7s$ directly adjacent to the $3s$ share the same polarity on the low-end of the interval, namely $[0, 7)$. All $7s$ are blocking $[0, 7)$ and there are 9 different colors required for all $3s$, but we only have 7 colors left in the interval from $[7, 14)$.

A positive instance of NAE-3SAT results in a positive instance of 3DS-IVC.

If the instance of NAE-3SAT is positive, then there is a variable assignment that is valid. We construct a solution of the created instance of 3DS-IVC out of the variable assignments of a solution of NAE-3SAT.

If v_1 is true, color the wire of v_1 to give it positive polarity. If v_1 is false, give the wire of v_1 negative polarity. This forces the coloring of all 7s in instance.

The only question left is "can we color the 3s?". That answer has to be true because we know the instance of NAE-3SAT is positive instance. Hence, for any clause that clause is valid and the 3 variables in that clause are not all equal. So at least one is true and at least one is false. Therefore all 7s in the clause object cannot have the same polarity. Two of the 7s share same polarity, and one has opposite polarity.

Assume 2 positive and 1 negative (without loss of generality). The 3 that is connected to the negative we will color $[0, 2)$. And the other two 3s we color with $[7, 9)$ and $[10, 12)$. That coloring is valid for that clause. we can color all 3s with a similar process.

If the created instance of 3DS-IVC is positive, then the instance of NAE3-SAT is also positive.

Since the instance of 3DS-IVC is positive, there is a valid coloring of the vertices of the 27-pt stencil. We infer the values for NAE-3SAT by looking at the polarity of the wire. If $(2i - 1, 2, 1)$ is colored with interval $[7, 14)$ then we set v_i to false. If it is colored with interval $[0, 7)$ then we set v_i to true.

If we were able to color the graph, then the triangle of 3s were colorable in 14 colors. And therefore for each clause, one of the three variables has a different value than the other two. This makes the NAE-3SAT instance a positive instance. □

Since the 3DS-IVC problem is in NP and is harder than NAE-3SAT which is an NP-Complete problem, we have the following result.

Theorem 3.4.3. *Deciding whether a 27-pt stencil can be colored with less than K colors is NP-Complete.*

Note that at this point, we do not know whether coloring a 9-pt stencil is an NP-Complete problem or not. Fundamentally, the reduction for 3DS-IVC works because the tube, wire, and triangle graph can be embedded in a 27-pt stencil. But that tube, wire, and triangle graph is not planar, so it can not be embedded in a 9-pt stencil. As such, the complexity of coloring the vertices of 9-pt stencil graphs with intervals remains open.

3.5 Heuristics

3.5.1 Greedy Algorithms

For the problem of coloring with intervals, we design greedy algorithms. We pick vertices one by one; When we pick vertex v, we give it the lowest color interval of width $w(v)$ that does not intersect with the color interval of one of the neighbors. To find such an interval, we first sort the color interval of neighbors by the lower end of the intervals. This enables to find the lowest color interval of length $w(v)$ that is available in a single pass over the neighbor colors intervals. This process has a complexity of $O(\Gamma(v)\log\Gamma(v))$ for vertex v. For the whole graph, the complexity of greedy coloring is $O(E\log E)$.

This greedy coloring has some upper bound on the number of colors used, even though it is higher than one would hope.

Lemma 3.5.1. *Any greedy coloring will color vertex v with an interval that ends at most with color* $\sum_{j\in\Gamma(v)} w(j) + (\Gamma(v)+1)w(v) - \Gamma(v)$

Proof. In the worst case, each neighbor uses different color intervals from one another, preventing $\sum_{j\in\Gamma(v)} w(j)$ colors from being used. When sorted, each of these color interval could be separated from the previous one (or from color 0) by exactly $w(v)-1$ colors. This forces the greedy algorithm to color v with an interval which starts after the one of all the neighbors at color $\sum_{j\in\Gamma(v)}(w(j)+w(v)-1)$. □

By this analysis, we know that the worst case is achieved when the algorithm colors the vertex of high weight after its neighbors have been colored with unfortunately spaced intervals. This leads us to design two broad categories of order in which to color vertices. Either you color early vertices/structures with high weights, or you color vertices in an order where vertices are not colored after all its neighbors (usually).

We describe first coloring in geometric patterns. The first one is to color vertices line by line (and then plane by plane in 3DS-IVC): we call this algorithm *Greedy Line-by-Line (GLL)*. The second one does not favor a particular dimension and orders the vertices using the recursive order Z-Order: we call this algorithm *Greedy Z-Order (GZO)*.

To color vertices based on the weight, the simplest ordering is simply to sort vertices in the order of non-increasing weights. We call this algorithm *Greedy Largest First (GLF)*.

From the analysis of the problem, we know that some structure of the instances are important, namely cliques and odd cycles. Since the clique of largest weight will be the structure which is likely to set the total number of colors, we designed an algorithm to color cliques first in non-increasing order of weight. Of course, there are multiple vertices in a clique and they are colored in an arbitrary order. It is also possible that some vertices of a clique have already been colored as part of a different clique; in this case, we follow the greedy principle and leave them untouched. We call this algorithm *Greedy Largest Clique First (GKF)*.

Note that we could pick the vertices in the clique in a particular, smarter, order. Since all the cliques in 2DS-IVC and 3DS-IVC are of constant size, we opt to try all the permutations of the vertices in the clique and only retains the permutation that leads to the best number of colors for that clique. This adds a $4! = 24$ overhead in the case of 2DS-IVC and a $8! = 40320$ overhead for 3DS-IVC. Since checking all $8!$ permutations per clique was too time consuming in our experiments, the algorithm implemented in the 3D cases was slightly modified from its 2D counterpart. Instead of examining all possible orders of a clique, we sorted the vertices inside the clique by non-increasing weights. We call these algorithms *Smart Greedy Largest Clique First (SGK)*.

3.5.2 Bipartite Decomposition

The 9-pt 2D Stencil and 27-pt 3D Stencil graphs we are interested in are very similar to bipartite graphs. We can use that property to design approximation algorithms for the 2DS-IVC and 3DS-IVC problem. We will explain the construction on 2DS-IVC and explain how the construction extends to other graph, including 3DS-IVC.

Here is how *Bipartite Decomposition* works. Consider individually each of the Y rows the 2DS-IVC instance. Each row is a chain of vertices, which is a bipartite graph and can be colored optimally using the algorithm presented in Section 3.3.2 in $\Theta(XY)$. Let $c(x,y)$ be the lower end of the color interval associated with vertex (x,y) in that coloring. And let $RC = \max c(x,y) + w(x,y)$ be the maximum color used by any of the rows. $RC \leq maxcolor^*$ is a lower bound of the optimal number of colors of the instance since it is the optimal coloring of a subgraph of the original instance.

Note that if we were to color vertex (x,y) with $start(x,y) = c(x,y)$ then the coloring would possibly be invalid since a vertex could share a color with one of its neighbors in the row above or the row below. *Bipartite Decomposition* colors vertex (x,y) with

$$start(x,y) = c(x,y), \forall x,y, y \equiv 0[mod 2]$$

$$start(x,y) = RC + c(x,y), \forall x,y, y \equiv 1[mod 2]$$

This can be done in $\Theta(XY)$ which makes *Bipartite Decomposition* an algorithm in $\Theta(XY)$.

That coloring is feasible since even rows are being colored using colors from $[0, RC)$ and odd rows are being colored using colors from $[RC, 2RC)$. Furthermore, the coloring uses at most $2RC$ colors. So, we have $maxcolor \leq 2RC \leq 2maxcolor^*$. In other words, we obtain the following theorem.

Theorem 3.5.2. *Bipartite Decomposition is a 2-approximation algorithm for 2DS-*

IVC.

The construction of *Bipartite Decomposition* works because once each row r has been colored, the row can be contracted into a single vertex of r of weight $w(r) = \max c(x,r) + w(x,r)$, and the resulting graph of the rows is a chain, which is bipartite itself. If one can decompose a graph G into p parts so that the contraction of G into p vertices is bipartite, and if the each part can be colored using a ρ-approximation algorithm, then *Bipartite Decomposition* can color G using at most $(2\rho)(maxcolor^*)$ colors.

In particular for 3DS-IVC, each layer of the graph can be colored with the 2-approximation algorithm for 2DS-IVC. Then the graph of the layer is a chain, which is bipartite.

Theorem 3.5.3. *Bipartite Decomposition is a 4-approximation algorithm for 3DS-IVC.*

Bipartite Decomposition by how it colors the vertices is really designed to be an approximation algorithm. It can lead to vertices using a high color interval without having neighbors using the most of the lower colors. We introduce a post optimization that recolors each vertex one at a time using a greedy principle. First, the vertices are listed as members of a K_4 (in 2D) or K_8 (in 3D). Next, all K_4 are sorted in non-increasing order by the sum total of their weights. Lastly, the vertices are sorted within their K_4 by increasing order of the lowest value in their scheduled interval. This produces an ordering of vertices that can be rescheduled one at a time. We call this algorithm *Bipartite Decomposition + Post (BDP).*

3.6 Experiments

3.6.1 Experimental Setting

The algorithms are written in Python and are interpreted using CPython 3.9.4. The machine that runs the code is equipped with an Intel i9-9900K and runs Windows 10.

When the experiments are run, no other workload runs on the machine at the same time.

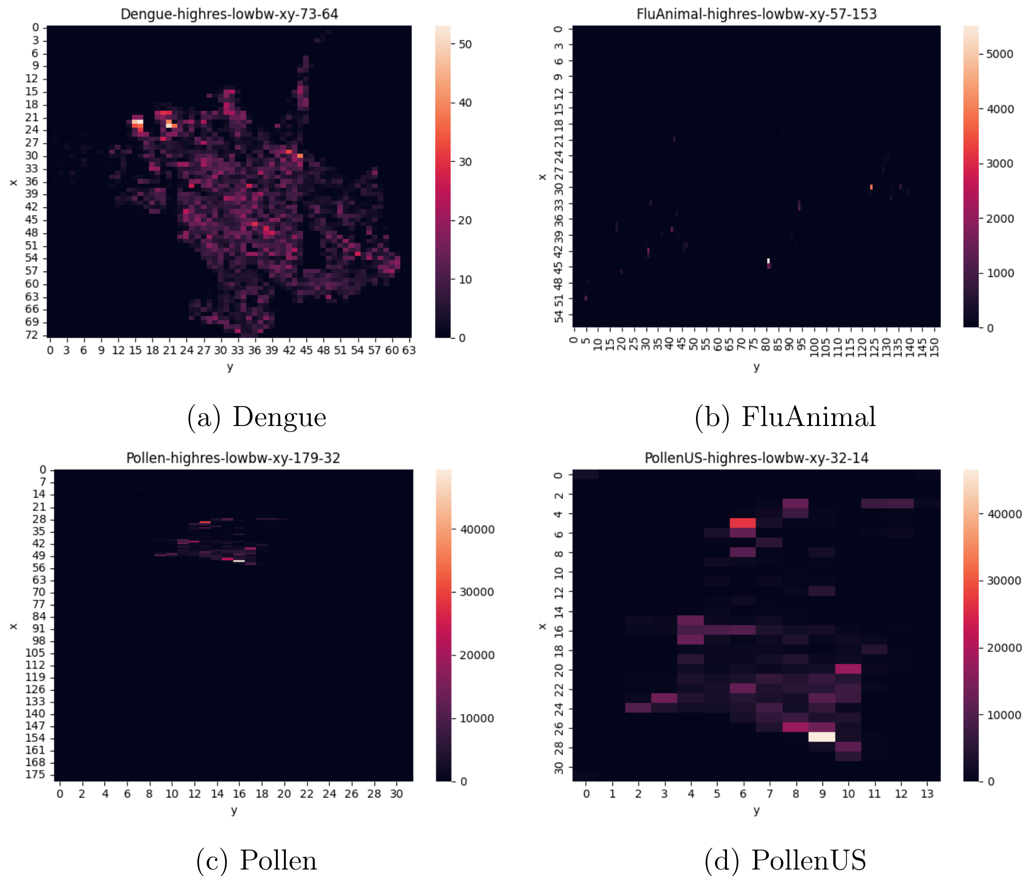
We obtained 4 datasets from the authors of [11]. Each dataset represents events located in space and time giving us a point in a $(lat, long, time)$ 3D space and is used to compute a voxelized kernel density of events. Each dataset can be analyzed under the light of different "bandwidth" which are distances within which an event can impact a voxel. For 2DS-IVC we project the dataset on each of the 2D plane: xy, xt, yt.

Each dataset is decomposed in a uniform 2D (or 3D for 3DS-IVC) grid composed of X columns and Y rows (and Z layers for 3DS-IVC). The possible values of X and Y are constrained by the bandwidth as the size of the region needs to be at least twice larger than the bandwidth. We list all powers of 2 for X, and Y (and Z for 3DS-IVC) as well as the largest value that can accomodate the bandwidth.

The first dataset is **Dengue** and comes from cases of the Dengue fever that occured in Cali, Colombia in 2010 and 2011. **FluAnimal** comes from the Animal Surveilllance database of the Influenza Research Database and contains an entry for each confirmed case of avian flu worldwide from 2001 to 2016. **Pollen** comes from geolocalized tweets mentioning keywords such as Pollen and Allergy between February 2016 and April 2016 by US users. **PollenUS** is a restriction of the Pollen dataset to the contiguous continental United States. Figure 3.4 presents a projection of each dataset on the xy plane for the largest partitioning that makes sense for the bandwidth. In total, there are 1587 instances of 3DS-IVC and 852 instances of 2DS-IVC.

3.6.2 2D Results

We used performance profiles to visualize the quality of heuristics. In these performance profiles, tau is the ratio between the value of $maxcolor$ produced by an algorithm to the number of colors obtained by the best algorithm for that instance. If the line for an algorithm goes through (tau, $Proportion$), then that algorithm is no



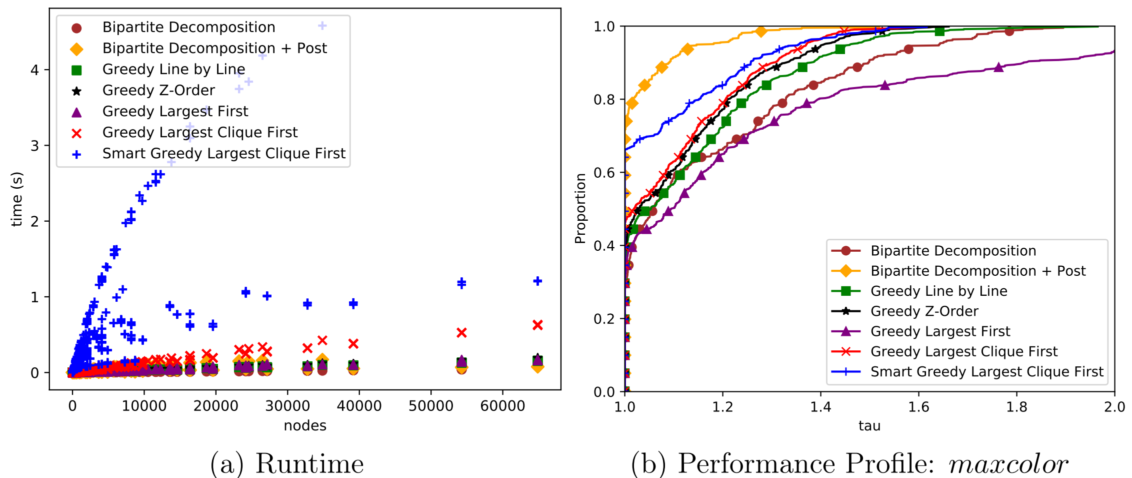
(a) Dengue (b) FluAnimal

(c) Pollen (d) PollenUS

Figure 3.4: Instance Samples

worse than *tau* times the best known solution on *Proportion* percent of the instances. The runtime comparison and performance profile for 2D instances can be found in Figures 3.5a and 3.5b, respectively. Performance profiles broken down by 2D dataset are shown in Figure 3.6.

In general, BDP performed substantially better than all other algorithms. On average, BDP obtained a solution within 1.03 times the lower bound of maximum K_4. BDP was 182% faster than SGK and required 1.69% less colors. BDP and SGK yielded the highest percentage of solutions that can be proven optimal with 58.7% and 63.3%, respectively. Although SGK obtained quality solutions, SGK was the slowest algorithm by a significant margin. SGK was anywhere between 160% and 182% slower than all other heuristics.



(a) Runtime (b) Performance Profile: *maxcolor*

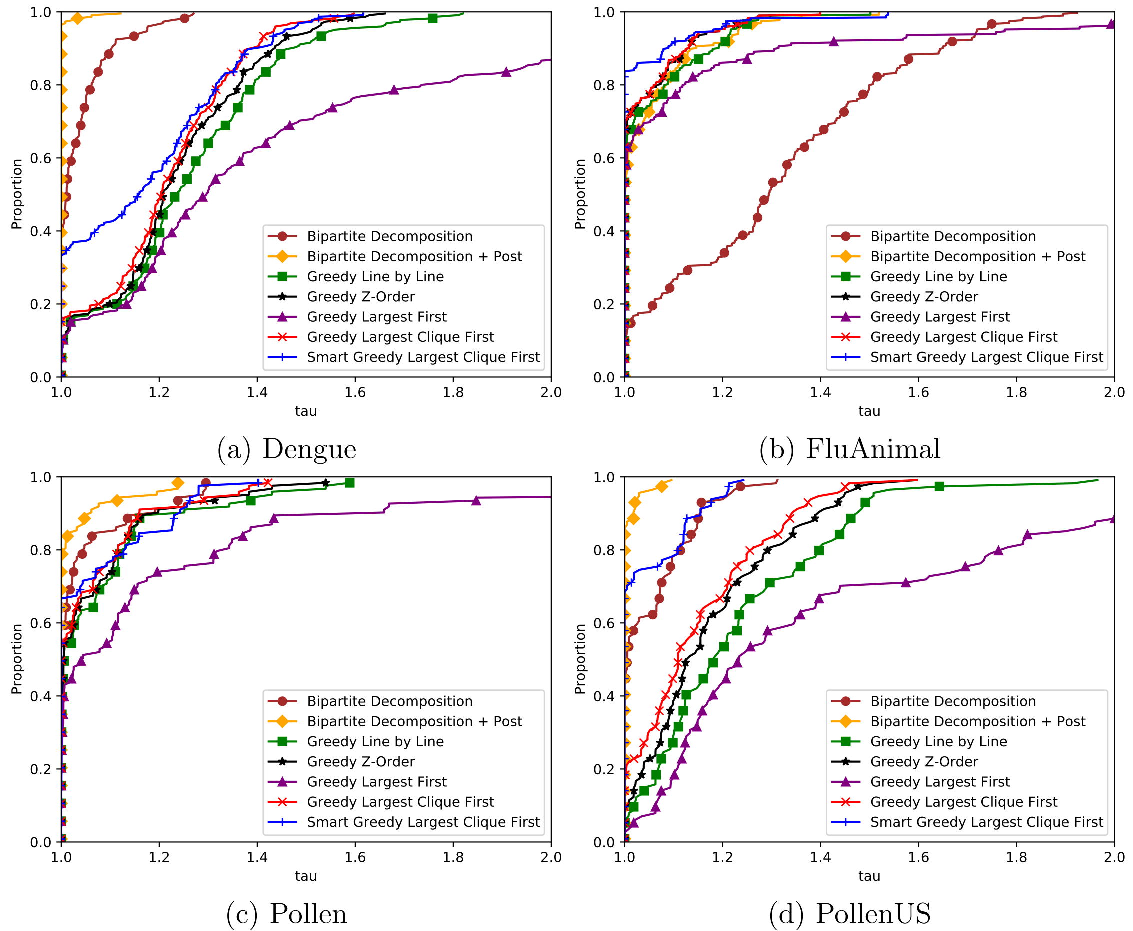
Figure 3.5: 2D Results (All Instances)

BDP obtained the best average *maxcolor* in all instances except FluAnimal. On these instances, SGK performed 8% better than BDP in terms of *maxcolor*, whereas BDP obtained a value for *maxcolor* similar to the other greedy algorithms. Overall the algorithms performed vastly different when compared with the other instances. This could be due to the fact that the instances of FluAnimal are very sparse. The performance profile for this particular instance can be found in Figure 3.6b.

The post processing associated with BDP improved the performance of the Bipartite Decomposition by 2.49%. Although this number may seem low, it was enough to establish BDP as the dominant heuristic in almost all 2D cases, whereas the original BD was merely average in performance. The post processing was 136% slower on average; however, this number may be skewed. In many cases, BD obtained a solution faster than it could be measured. Thus, the wallclock used to measure time returned a value of 0.

3.6.3 3D Results

The runtime comparison and performance profile for 3D instances can be found in Figures 3.7a and 3.7b, respectively. Performance profiles broken down by 3D dataset are shown in Figure 3.8.



(a) Dengue (b) FluAnimal

(c) Pollen (d) PollenUS

Figure 3.6: Performance Profile for 2DS-IVC: *maxcolor* broken down per dataset

GLF and SGK were the clear winners on 3D instances. SGK was marginally better than GLF, yielding less than a 0.57% improvement in average *maxcolor* and finding optimal solutions in 11.8% more instances. However, GLF was significantly faster. GLF had a runtime 142% faster than SGK, 128% faster than BDP, and 120% faster than GKF. SGK was the slowest algorithm by a sizeable factor. SGK was 25.3% slower than BDP, 38.9% slower than GKF, and 154% slower than GLL.

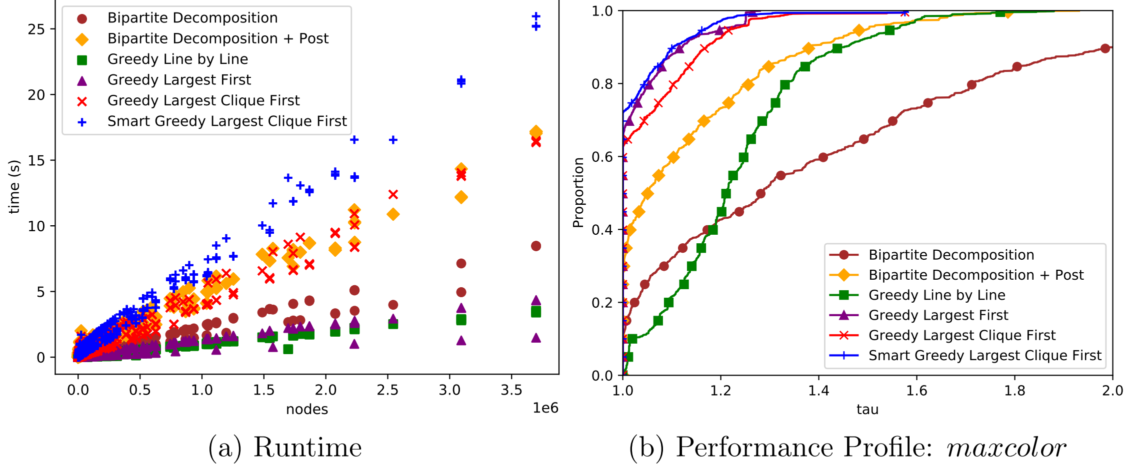
BDP had a mediocre performance on 3D instances, whereas it was the clear favorite on the 2D instances. In 3D, BDP obtained an average *maxcolor* with a higher than average runtime. Furthermore, the different 3D instances seemed to have a greater impact on algorithm performance than in the 2D cases.

Considering the addition of the z-axis, it is likely that vertices, which are consecutive in the sequence of largest weights, will be located on different planes. If this is the case, then the planes that separate them effectively function as layers of insulation. This allows the set of colors initially assigned to the large weighted vertices to remain 0-aligned throughout the greedy algorithm. Consequently, a lower *maxcolor* can be achieved because the remaining intervals can be tightly packed.

We would also expect to see the 2D results upheld in instances where consecutive vertices in the sequence of largest weights appear on the same plane. The results seem to reflect this argument: 18.1% of 3D instances BDP performs strictly better than SGK in terms of *maxcolor*. We conclude that specific distributions of weights will be advantageous to different algorithms. Hence, the construction of different instances can explain the disparity between the 2D and 3D results.

3.6.4 Optimal coloring based analysis

In order to further analyze the performance of the heuristics, we designed a Mixed Integer Linear Program (MILP) and solved optimally as many instanes as we could. We solved the MILP using Gurobi and let the solver run for one day per instance on a node of a computing cluster. Most of the instances were solved with a provably



(a) Runtime

(b) Performance Profile: *maxcolor*

Figure 3.7: 3D Results (All Instances)

optimal solution within a day: Only 21 (2.46%) 2D instances and 269 (16.9%) 3D instances were not solved.

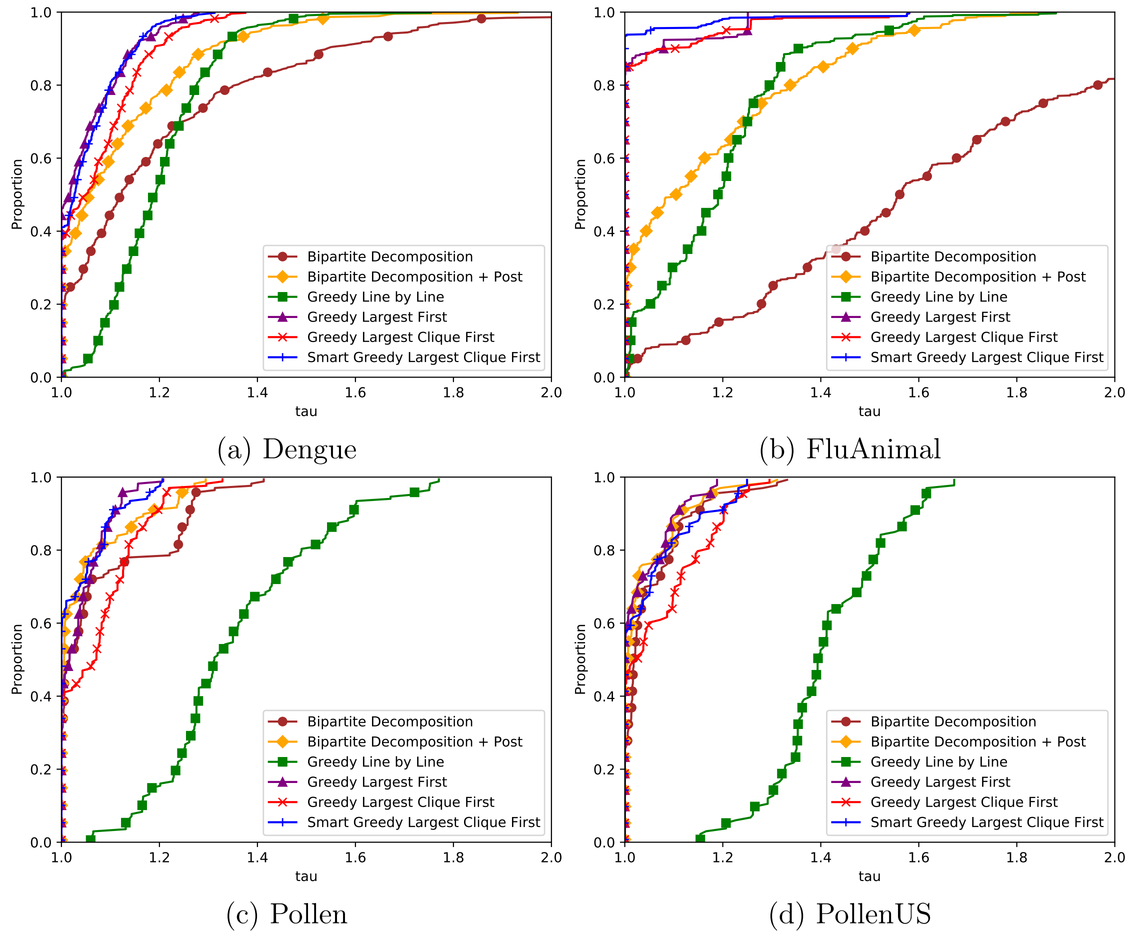
We replotted performance profiles for both 2D and 3D instances that were solved by the MILP in Figures 3.9a and 3.9b, respectively. These new figures are virtually the same as the original performance profiles. This indicates that for most instances, one of the heuristics had found an optimal solution or a near optimal solution.

Having optimal solutions also enable us to study the quality of the max clique lower bound. The max clique lower bound was different than the optimal solution value in only 57 (4.33%) of the 2D instances and 22 (2.65%) of the 3D instances. Furthermore, in the instances where they differed, the difference was less than 0.01%.

It is important to remember that not all instances were solved optimally by the MILP solver. Therefore, it is possible that these unsolved instances are more complex. These unsolved instances may exhibit a greater difference between the max clique lower bound and the optimal solution.

3.7 Coloring for Space Time Kernel Density Estimation

To validate the model and approach on a real application, we obtained the STKDE code used by the authors of [11]. In this application, some events (points) are located in a 3D space and the space is discretized in voxels. The computational load is



(a) Dengue

(b) FluAnimal

(c) Pollen

(d) PollenUS

Figure 3.8: Performance Profile on 3DS-IVC: *maxcolor* broken down by dataset

carried by the points which provide contribution to the voxel it is in and nearby voxels within a particular radius called the bandwidth. A more precise description of the application is given in [11].

The application has many modes of parallelization but we focus on the strategy that partitions the points spatially in boxes no smaller than twice the bandwidth. The points in a box are processed in a single (sequential) task and two neighboring boxes can not be processed simultaneously.

The problem of finding the best ordering of the tasks boils down to the 3D 27-pt stencil coloring problem that we consider in this manuscript where the weight of a task is given by the number of points contained in that box. We modified the application to call our coloring algorithm and then used OpenMP's tasking construct to create the parallel execution. The OpenMP tasks are created in order of increasing start of their color interval with dependencies to the neighboring boxes. So this creates a DAG of tasks managed by the OpenMP runtime which is a 27-pt stencil with edge oriented in a fashion compatible with the coloring.

We took 6 of the instances and parameters that were reported to take more than 1 second of total runtime in sequential execution in [11]. We executed the application on a machine equipped with an Intel Core i5-11600K which is a 6-core (12 hyperthreads) processor and 32GB of memory. The machine runs Debian 11 with a Linux kernel in version 5.10 and the code is compiled with GCC 10.2.1. Each code is run 5 times and the reported times are averaged across the 5 run. We only report the computation time and not the time to perform input/output, data preparation, and coloring.

Figure 3.10 shows the relation between the number of colors in the coloring and the time the application took to compute. Every case shows a linear correlation between colors and runtime, although that correlation is weak in two of the cases. This confirms that modeling the application as a coloring problem on a stencil makes sense.

Although on `PollenUS-veryhighres-lowbw`, the difference between the best and the worst color is 38%, the difference in runtime is only 4%. This is because the maximum color in the schedule does not directly relate to runtime. In fact, the maximum number of colors indicate the length of the critical path in the graph of tasks scheduled by the OpenMP runtime. And in that case despite the length of the critical path decreased by 38%, that critical path represents only 5% of the total work of the application.

The highest decrease in time happened on `FluAnimal-highres-highbw-3d-16-16-32` where the best time is 27% lower than the worst time. The worst time is achieved by the worst coloring which induces a critical path of 10% of the work.

It is also worth noting that we quantify the weight of the tasks in term of number of points. But the runtime bottleneck of the application in the architecture is the memory subsystem which is shared among the cores. So as long as enough cores are working to saturate the memory subsystem, the performance may not suffer even if a few cores are idle.

The BD and BDP coloring algorithms induce the same Parallel Task Graph in the OpenMP runtime since the BDP coloring is just a compaction of the BD coloring. But, in practice, the BD and BDP algorithms can yield different performance. We believe that the root cause is that despite the underlying task graph is the same, the tasks are given to the runtime in a different order. And that could impact the scheduling decisions made by the OpenMP runtime.

3.8 Conclusion

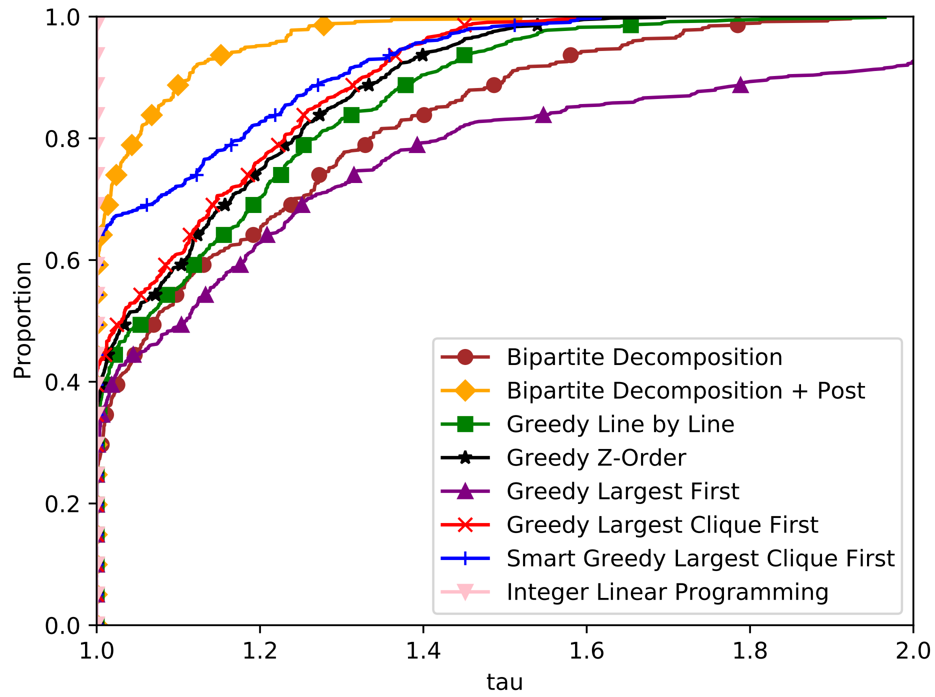
We investigated the problem of interval vertex coloring of 9-pt and 27-pt stencil graphs. We showed that the 5-pt stencil and 7-pt stencil relaxations of the problem can be solved in polynomial time. We also proved that the decision problem on 27-pt stencil is NP-Complete.

Furthermore, we proposed heuristics with very good performance in both the 2D

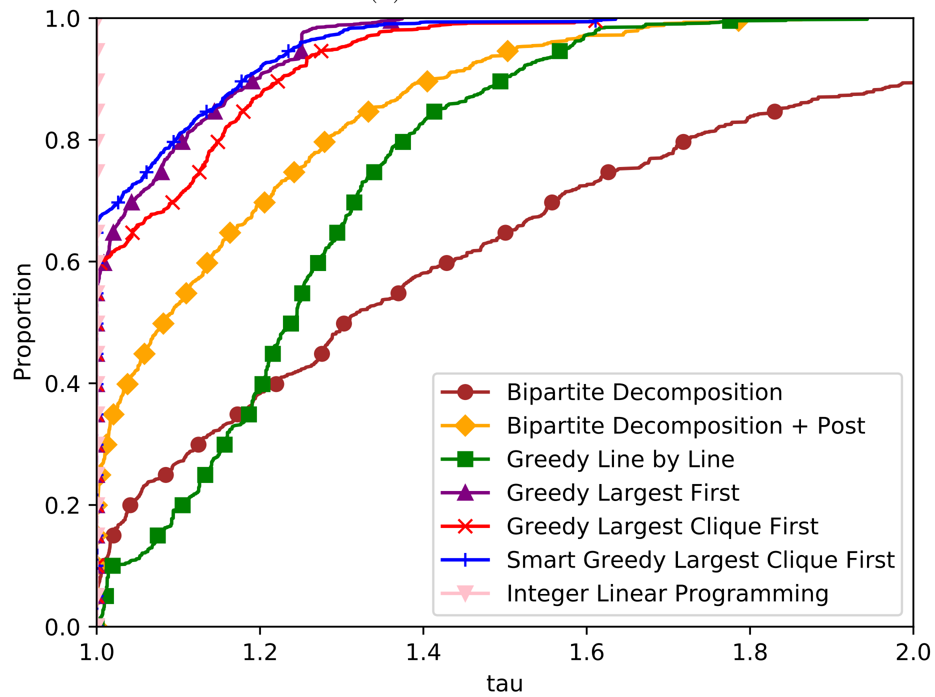
and 3D variants of the problem. The Bipartite Decomposition + Post (BDP) heuristic is an approximation algorithm which performs exceptionally well in nearly all 2D cases. In the 3D cases, the Smart Greedy Largest Clique First (SGK) algorithm obtained the highest quality solution overall, but the Greedy Largest First (GLF) algorithm achieved a similar quality of solution in a significantly shorter runtime.

Using an ILP, we were able to show that the heuristics with good performance are optimal or near-optimal in many cases. We also integrated our heuristics in a real stencil application showing that better coloring will improve runtime performance.

Some open problems remain. Is the problem of interval coloring 9-pt stencil graph NP-Complete or polynomial? Can we design approximation algorithms for coloring 27-pt stencil with an approximation ratio better than 4?



(a) 2D Instances



(b) 3D Instances

Figure 3.9: Performance Profiles with ILP

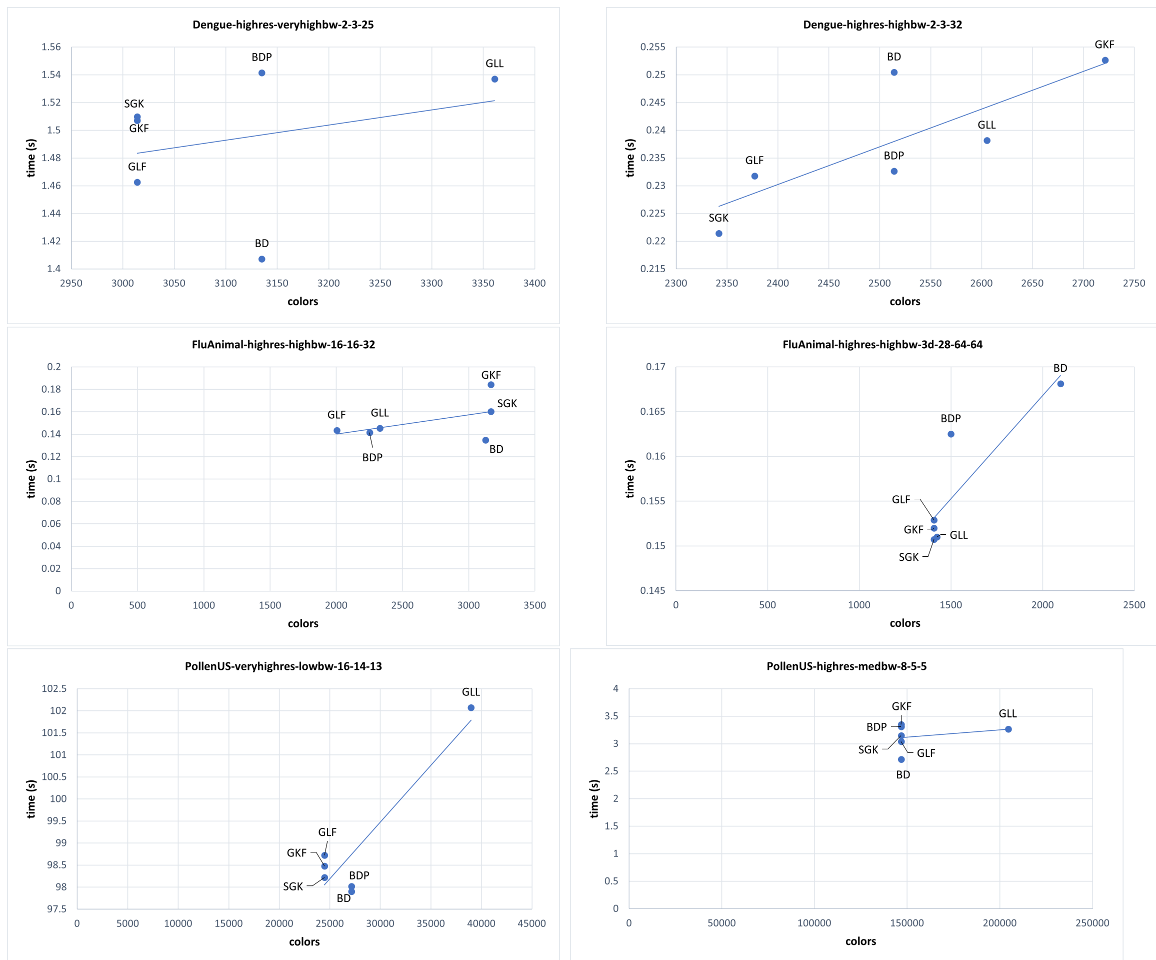


Figure 3.10: Scatter plot of number of colors and execution time of the STKDE application. Each scatter plot presents different coloring algorithm. A linear regression line shows positive corelation between number of colors and runtime in all 6 cases.

CHAPTER 4: DATAFLOW ALGORITHMS

4.1 Introduction

Graphs are a key mathematical object of modern science as they are used to model a variety of objects of studies including physical objects, roads, computer networks, and social interactions. With the increase in complexity of studies that are performed, the size of graphs that are used has increased. Consequently, computational costs of analyses have increased and the machines we use to perform these calculations have grown more parallel and more distributed.

Executing graphs on parallel and distributed machines is complicated because of the irregularity of the memory access patterns. Simple solutions involve using some form of a locking mechanism. However, the locking overhead usually dominates the calculations. Some problems admit optimistic algorithms where race conditions are ignored at first and the solution is later examined and fixed. It is possible that a race condition happens in a way that leads to an incorrect solution [36, 37]. These optimistic algorithms fundamentally require a later reconciliation phase, which can be as costly as the algorithm itself.

A third category of dataflow graph algorithms rely on a partial order to the graph, where vertices are processed in an order compatible with that partial order.

The most classic dataflow algorithms are Luby's algorithm for Maximum Independent Set [38] and the Jones-Plassmann algorithm for graph coloring [39]. However, many problems admit dataflow algorithms, such as maximum cardinality matching [32]. These algorithms are particularly suitable for the setting where each vertex is its own independent computational node. They are also easily written in think-like-a-vertex programming models [40]. They have also been used on shared memory

systems, including the Cray XMT [32, 36].

A bottleneck to the execution of these algorithms is the longest chain of vertices set by the partial order. Since the precise partial order often does not matter for correctness, dataflow algorithms often use a random order. Random orders can be generated in a distributed way and have been shown to yield desirable properties in several types of graphs [38, 32]. In this chapter, we investigate alternative ways to generate random orders that minimize the length of the longest chain of vertices in the algorithm, minimizing its runtime.

4.2 Problem Statement

4.2.1 Dataflow Algorithms

All dataflow graph algorithms share a similar structure. We explain in detail how Luby's algorithm computes an Independent Set [38] as a distributed algorithm. Let $G = (V, E)$ be a graph. We denote the neighbors of vertex v by $\Gamma(v)$, the degree of v by $\delta(v) = |\Gamma(v)|$, and the maximum degree in the graph by $\Delta = \max \delta(v)$.

In Luby's algorithm, each vertex v starts by picking a random number $r(v)$ uniformly in $[0, 1)$, which is assumed to be unique, and sends it to each of its neighbors. Vertex v notes which of its neighbors u have the property that $r(u) < r(v)$.

Vertex v marks its own state as `unknown`. It awaits a message from each of its neighbors u if $r(u) < r(v)$ which contains the state of neighbor u. If u's state is `marked`, v changes its state to `unmarked`.

After receiving messages from all neighbors u, if vertex v state is `unknown`, it changes its state to `marked`. And finally v sends its state to all its neighbors u such that $r(u) > r(v)$. At the end of this process, all vertices in the `marked` state are a maximal independent set (by inclusion).

Other dataflow algorithms are similar in structure. A random number is assigned to each vertex. And each vertex executes an algorithm only after all of its neighbors with a lower value have been executed.

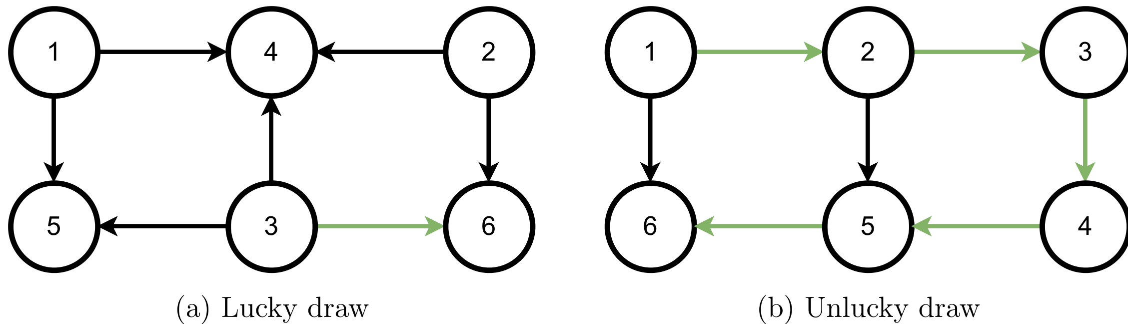


Figure 4.1: The execution time of dataflow algorithm depends on the random number generation. Black edges highlight the order of the dataflow, while green edges show the critical path. On a lucky draw, the critical path of the algorithm contains only 2 vertices, while an unlucky draw can have 16 vertices in its critical path.

In Luby's algorithm, each vertex v executes an algorithm of complexity $\Theta(\delta(v))$, linear in its number of neighbors. However, since each vertex has to wait for some of its neighbors to complete, the entire process might not unfold in $\Theta(\Delta)$, proportionally to the maximum degree of the graph. The time it takes for the algorithm to unfold depends on the longest chain of communication induced by the algorithm, which depends on the random numbers generated.

A grid graph is used as an example in Figure 4.1. In a lucky draw (Figure 4.1a), random numbers are generated in a way that leads to lots of parallelism. The black arrows show the direction of the communication in the graph. The green arrows show the longest chain in the oriented graph. In this case, all chains are of two vertices and an arbitrary one was highlighted.

In an unlucky draw (Figure 4.1b), the random numbers are generated in a way that leads to no parallelism as the longest chain in the oriented graph visits every vertex in the graph. Of course, both of these draws are extremely unlikely to happen.

4.2.2 Combinatorial Optimization Model

Ultimately, dataflow algorithms rely on the underlying coloring of a graph with intervals. Let $G = (V, E)$ be the graph that gets processed by the dataflow algorithm. Let w be a weight function on vertices such that $w(v)$ is the processing time of an

algorithm for vertex v. The dataflow algorithm is dependent upon two neighboring vertices $(x, y) \in E$ to not be executed simultaneously. It assumes they run at times $[start(x), start(x)+w(x))$ and $[start(y), start(y)+w(y))$ such that these two intervals are disjoint. This is the definition of a coloring of the graph with intervals of length given by the weights of the vertices. The objective is to minimize the total number of used colors, which is equivalent to minimizing the total runtime of the application.

How good the model is will largely be determined by how the weight function w is set. On a computing machine with substantially larger latency than execution time at each node, a good model will be achieved by setting all w to 1 and solving the standard coloring problem. If the processing at each vertex is the primary cost of the dataflow algorithm, then setting $w(v)$ to the complexity of the vertex algorithm for each vertex is the right call. For most algorithms, vertices will need to gather partial information from their neighbors, do some processing, which is usually proportional to the number of neighbors, and finally, communicate the partial information to the neighbors. All operational costs will be proportional to the number of neighbors of a vertex. Setting the weight to the degree of the vertex $w(v) = \delta(v)$ often makes the most sense. We will make this assumption going forward even though the analysis can be adapted to other weight functions.

The distributed dataflow algorithm solves that problem using a particular ordering algorithm. But fundamentally, any coloring of the graph with intervals would be sufficient to derive a correct execution of the dataflow algorithm. And better colorings would lead to better runtimes for the execution.

In the context of a distributed graph, it makes sense to run a distributed coloring algorithm: aggregating the graph to a single computing node to execute a one-node algorithm would likely be prohibitively expensive compared to the rest of the dataflow algorithm execution time.

4.3 Deriving better partial orders

4.3.1 Methods

Luby's algorithm and most dataflow algorithms generate random numbers uniformly in $[0, 1)$ to derive the order of vertices. This yields a partial order in a distributed setting so each vertex v performs only $\Theta(\delta(v))$ calculations and $\Theta(\delta(v))$ communications. The communication term is required for a vertex to know its place in the order relative to its neighbors. At the level of the system, there are only $\Theta(E)$ calculations and communications. We call `Uniform` this particular ordering algorithm.

While that algorithm has optimal cost to derive a partial order, it may not derive the best order. In particular, this ordering does not leverage any properties of the graph and its vertices. We know from literature that coloring heuristics can benefit from considering vertex properties and local structure. In particular the Smallest Last [21] and Largest First [20] orderings are know to be good for the classic coloring problem [41]. Since Smallest Last is a dynamic ordering, it is costly to replicate in a distributed setting. Instead, we focus on emulating Largest Degree First.

Instead of generating random numbers uniformly in $[0, 1)$, we propose adjusting random number generation based on the property of the vertex drawing the number. In particular, we call `Linear` the algorithm where vertex v draws a number uniformly in $[0, \delta(v))$. This algorithm has the same complexity as `Uniform` but it generates an ordering that will tend to put vertices with high degree towards the end of the ordering.

However, a vertex v is guaranteed to be after all vertices u such that $\delta(u) = \delta(v) - 1$ only with probability $\frac{1}{\delta(v)}$ (for an infinite number of such vertices u). `Linear` is a good approximation of Largest First for vertices with dramatic difference in degrees. But it is a poor approximation of that ordering for graph with very large maximum degrees and many vertices of large degrees.

We suggest a third generation algorithm called `Exponential` where vertex v draws

a random number in $[0, 2^{\delta(v)})$. This algorithm still has the same communication and computational cost with an additional benefit: the probability that a vertex v has a random number $r(v)$ greater than all vertices u such that $\delta(u) = \delta(v) - 1$ is greater than $\frac{1}{2}$. Therefore, it is a better approximation of the Largest Degree First ordering.

4.3.2 Basic analysis

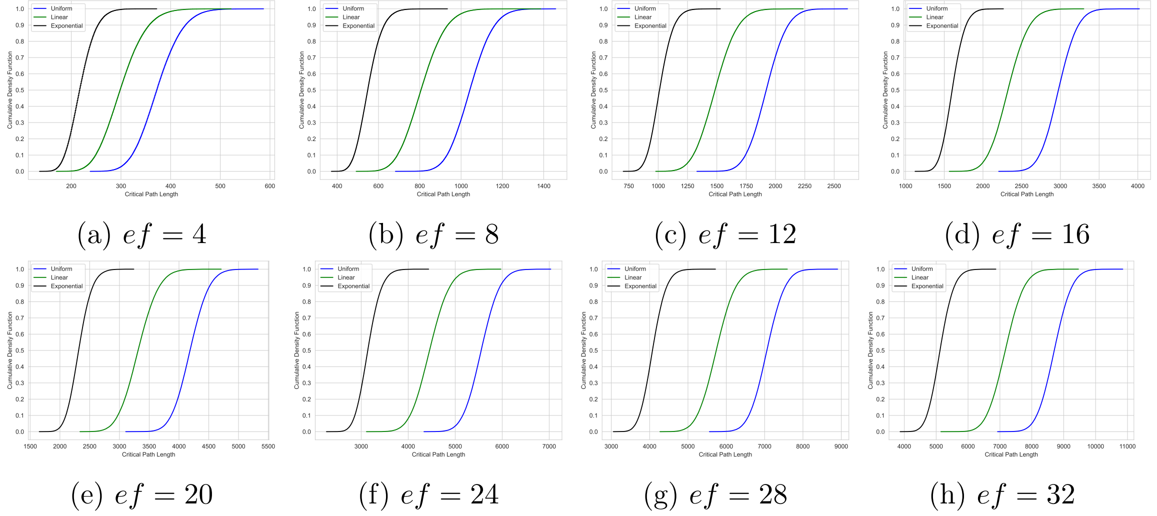
Regular graphs have the property that all vertices have the same degree. This category of graphs encompasses many typical structures. Cliques, cycles, torus (2d, 3d, or arbitrary dimension) are all regular graphs. Because all vertices have the same degree, all three algorithms behave in exactly the same way. Although each method generates random numbers in different intervals, all vertices in that method generate numbers in the same interval. Consequently, all three algorithms behave in the same way.

Star graphs show why the methods work differently. Consider a star graph of V vertices. The center vertex (hub) has a degree of $V-1$, while all other vertices (spokes) have a degree of 1. There are only two possible solutions (excluding symmetries) that can be generated by the distributed algorithm. Either the hub vertex is in between two of the spokes, or it is not. It does not matter if the hub is before all spokes or after all spokes. If the hub vertex is between the two spokes, the longest chain has 3 vertices; otherwise, it has 2 vertices.

`Uniform` will put the spoke vertex first with probability $\frac{1}{V}$, and will put it last will probability $\frac{1}{V}$. As $V \to \infty$, the algorithm will generate a path of 2 vertices with probability $\frac{2}{V}$ and a path of 3 vertices with probability $\frac{V-2}{V}$.

On the other hand, `Exponential` will force all spoke vertices have random numbers in $[0, 2)$, while the hub vertex will be a random number in $[0, 2^{V-1})$. The hub will have a random number greater than 2 with probability $\frac{2^{V-1}-2}{2^{V-1}}$. In all cases, the longest chain will be of 2 vertices with a probability greater than $\frac{2^{V-1}-2}{2^{V-1}}$, which goes to 1 as V goes to infinity.

In the case of the `Linear` algorithm, the spokes have random numbers in $[0, 1)$, while the hub has a number taken in $[0, V - 1)$. The hub has a random number greater than 1 with probability $\frac{V-2}{V-1}$. And so, the probability of having a path of 2 vertices tends to 1 as V approaches ∞.



(a) $ef = 4$ (b) $ef = 8$ (c) $ef = 12$ (d) $ef = 16$

(e) $ef = 20$ (f) $ef = 24$ (g) $ef = 28$ (h) $ef = 32$

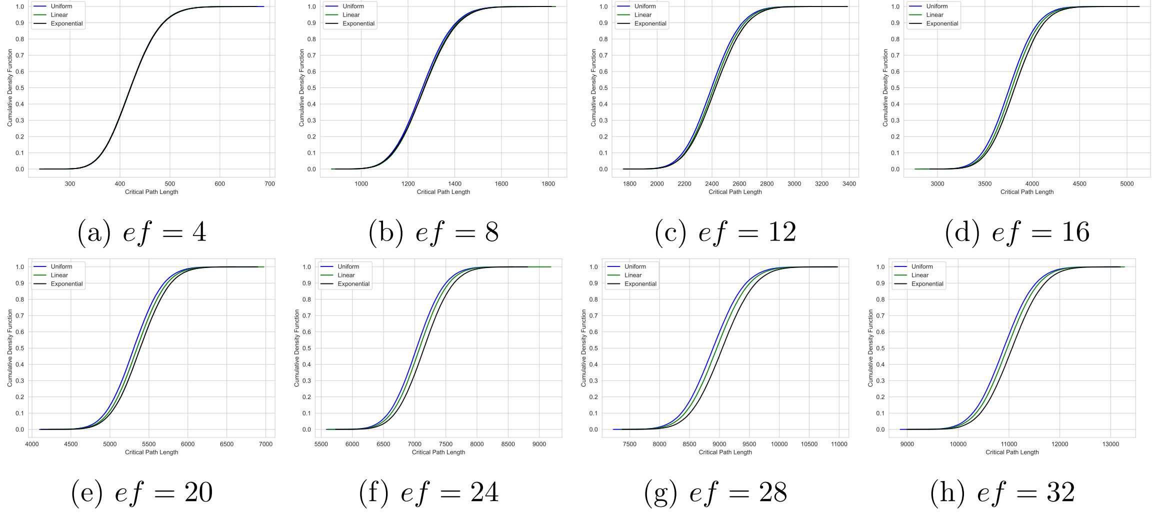
Figure 4.2: Cumulative Density Function of the longest chain induced by `Uniform`, `Expnential` and `Linear` on RMAT Graph with a = 0.10, b = 0.20, c = 0.50, d = 0.20 for different values of edge factor ef. The different values of edge factor show almost identical patterns for the length of the critical path.

Although star graph are not commonly found in real-world applications, many graphs, such as social networks, are similar to star graphs: they are structured like onions with dense center regions and layers of ever lesser dense regions. We believe that an algorithm like `Exponential` favors shorter paths in social networks because once a path enters a denser region is entered it tends not to exit it.

4.4 Study on Recursive Graph Model (RMAT)

4.4.1 Methodology

RMAT graphs have 2^n nodes. They are constructed by recursively splitting a square matrix into 4 quadrants: a, b, c, d. Each quadrant has an associated probability that a given edge will fall into that quadrant, so $a+b+c+d = 1$. Edges are generated one at a time and placed in a quadrant following the given probabilities, and recursively



(a) $ef = 4$ (b) $ef = 8$ (c) $ef = 12$ (d) $ef = 16$

(e) $ef = 20$ (f) $ef = 24$ (g) $ef = 28$ (h) $ef = 32$

Figure 4.3: Cumulative Density Function of the longest chain induced by `Uniform`, `Expnential` and `Linear` on RMAT Graph with a = 0.42, b = 0.19, c = 0.19, d = 0.02 for different values of edge factor ef. The different values of edge factor show almost identical patterns for the length of the critical path.

until the edge is placed in a 1×1 sub matrix. The number of edges in a graph is usually controlled by setting an edge factor ef which will generate $ef * 2^n$ edges.

RMAT graphs have several desired properties of many real graphs. They have a power-law degree distribution that resembles real graphs in several applications. They also exhibit a community structure and have a small diameter [42]. RMAT graphs are the 2×2 special case of Kronecker graph [43]. Note that RMAT is a directed graph model, however we need an undirected graph, so the matrix is symmetrized after generation.

All the studies that we conduct on RMAT graphs make the same assumption. We assume that our graph is derived from an underlying RMAT distribution, and we try to ascertain whether `Uniform`, `Linear`, or `Exponential` would obtain a shorter longest chain. Since RMAT is a probabilistic model and these methods themselves are randomized, statistical evidence is required. For an RMAT parameter set and an algorithm, we estimate the probability density function by sampling 1000 RMAT graphs generated by these parameters, and for each graph that was generated, by

sampling 100 executions of each ordering algorithm. This gives us $100k$ values of longest path length for each ordering for a particular set of parameters.

In general, we present the sampled Cumulative Density Function (CDF) of the longest chain and the confidence intervals for the expected length of the longest chain. We validated the significance of the difference in the expected length of longest chains between two orderings with a pairwise two-population z-test. The p-value was always substantially lower than 0.05 which indicates that all results presented on RMAT graphs are statistically significant.

4.4.2 Initial Investigation

As there seem to be no real consensus on what RMAT parameters to use to benchmark algorithms, we started our exploration by considering two sets of RMAT parameters that appear frequently in the literature. We have four initial questions. Does the ordering method make a difference? Does one of the methods lead to shorter path? How does edge factor impact the results? Do the parameters a, b, c, and d make a difference?

We used parameters $(0.10, 0.20, 0.50, 0.20)$ and $(0.42, 0.19, 0.19, 0.02)$. We varied the edge factor between 4 and 32. We present the Cumulative Density Function of the length of the longest path in Figures 4.2 and 4.3.

The statistic tests showed that the distribution are statistically significantly different. The difference on the RMAT parameters $(0.1, 0.2, 0.5, 0.2)$ was fairly important. It seems that for all edge factors, `Linear` leads to shorter longest paths than `Uniform` and `Exponential` leads to shorter longest paths than both of them. However, the difference on the RMAT parameters $(0.42, 0.19, 0.19, 0.02)$ was very small. (Even though it was statistically significant.)

The edge factor seems to have little impact on the relative performance of each method. The difference between orderings appears to be more pronounced for larger edge factor; however, the trends remained the same.

4.4.3 Exploring the RMAT Parameter Space

Our initial investigation revealed that the a, b, c, and d parameters are important, while edge factor did not appear to be important. `Exponential` seems to lead to shorter path than `Linear`; and `Uniform` seems leads to longer path. The question is whether the parameters we used were odd cases or whether the trends hold for any RMAT graph.

In this section, we explore the RMAT parameter space in a systematic fashion. We generated a set of 65 different parameters on graphs of with 2^9 vertices and fixed the edge factor to 16. We selected values of a in the range $[.3, .8]$ regularly. Subsequently, other parameters were selected as a fraction of the remaining number of edges. The remaining cases were seperated into $b > c$ and $b = c$. The precise values of a, b, c, d that we used are included in Table 4.1.

Table 4.1 also provides the confidence interval of the average length of the longest chain for all three orderings. It also provides the ratio of the average longest chain path between `Uniform` and `Exponential`, between `Uniform` and `Linear`, and between `Linear` and `Exponential`. We highlighted in bold the ratios that are greater than 1.15.

Since none of the ratios were smaller than 1 and all results were statistically significant, it does seem that on RMAT graphs, `Exponential` is better than `Linear`, which is itself better than `Uniform`. `Exponential` leads to path less than half the length of `Uniform` path on average with RMAT parameters $(0.30, 0.49, 0.08, 0.13)$. Overall, `Exponential` obtained a path at least 15% better than `Uniform` on 17 different RMAT parameters.

While it is unclear how the different parameters control for the difference in path length, it appears that smaller values of the a parameter seem to favor the `Exponential` ordering.

4.5 Real Graph Study

While it is encouraging that the `Exponential` ordering leads to shorter longest path on RMAT graphs, it does not necessarily hold that the result will be the same given graph extracted from real-world applications. We tested several real-world graphs from the Stanford Network Analysis Project (SNAP): CA-HepPh, Email-Enron, p2p-Gnutella04, roadNet-PA, soc-Epinions1, soc-pokec-relationships, web-Google, and WikiTalk. A summary of the properties of these graphs are given in Table 4.2. All these graphs have small world properties, except the graph of the roads of Pennsylvania, which is almost a regular graph. We included that graph as a control.

We present the summary statistics of the orderings in Table 4.2 and the Cumulative Density Function of the length of the longest path in Figure 4.4.

All the results are statistically significant. On two graphs, `Exponential` does not lead to the smallest longest chain in average: ca-HepTh and roadNet-PA. Although the difference in distribution is fairly small and the average length only differs by less than 2%.

On the other graphs, `Exponential` leads to longest chains shorter than `Uniform` by more than 7% in average, and by more than 15% on 4 of the graphs. Surprisingly, the average longest chain generated by `Uniform` is almost 4.5 times longer than the average longest chain generated by `Exponential`. `Linear` overall, sits in between `Uniform` and `Exponential`.

While we expected to see `Exponential` lead to much shorter longest chains than `Uniform`, it is not clear yet to why ca-HepPh does not follow the same trend. We hypothesize that ca-HepPh has one large cluster of vertices which is mostly completely connected. And as such, behaves in practice similarly to a clique.

4.6 Conclusion

In this chapter, we investigated the performance of distributed dataflow graph algorithms. We modeled the problem of optimizing the critical path of the partial order used by the algorithm using a formulation as a coloring problem with intervals of colors. We proposed two alternative ways to derive a partial order, `Exponential` and `Linear`. These methods rely on local properties of the vertices, which enable these orderings to run with no additional cost.

We investigated the efficacy of these algorithms on a large number of RMAT graphs. We showed that `Exponential` outperforms the state of the art on all tested RMAT parameters. We also tested the `Exponential` algorithm on 8 real-world graphs and showed it never loses more than 2% to state of the art and reduce the longest chain by more than 20% on 4 of these graphs.

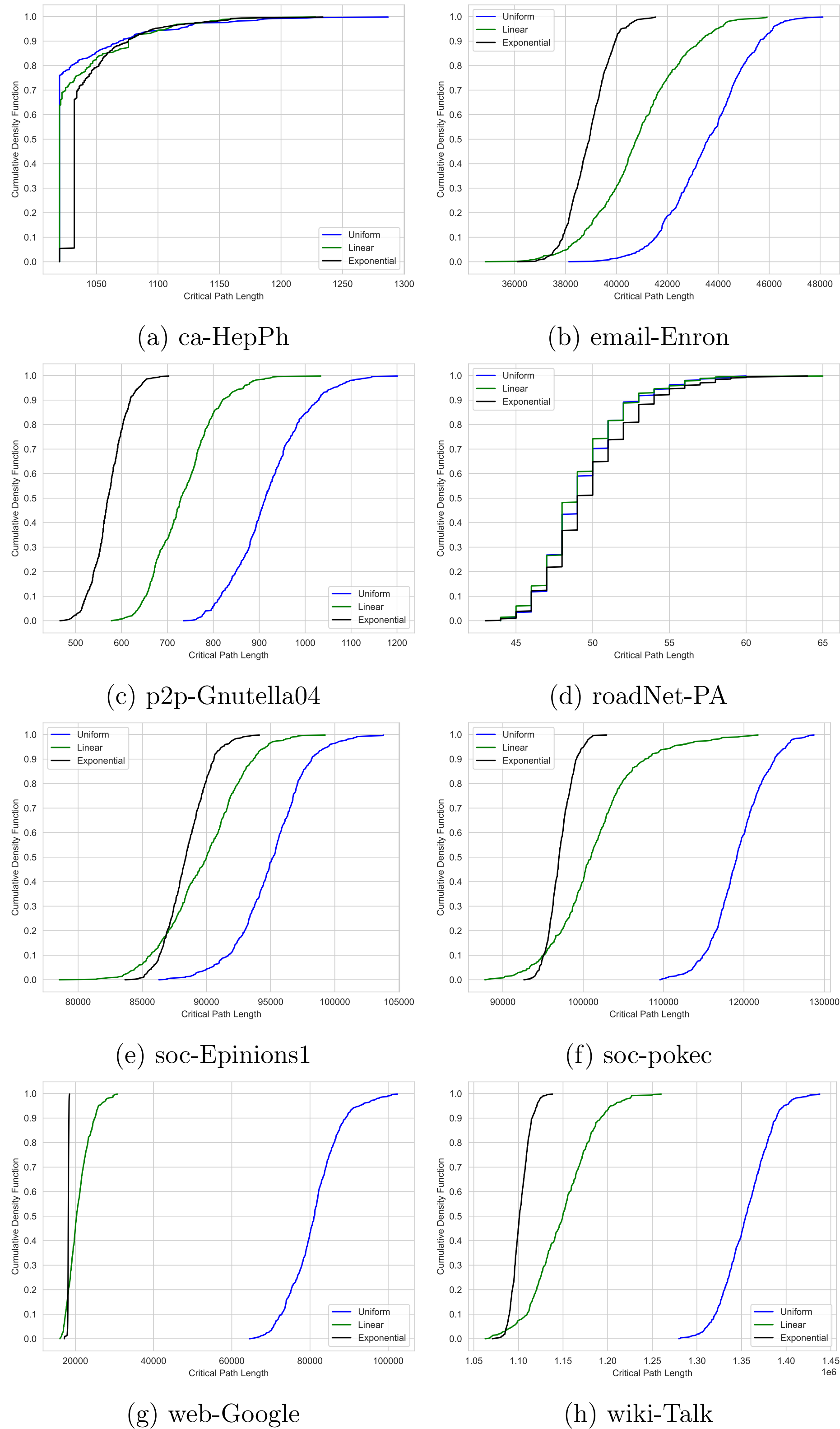
We would like to understand more precisely why `Exponential` is better than state of the art; we believe that investigating the behavior of the algorithm relative to the k-core decomposition of the graph might yield more insight. Finally, we also want to experimentally measure how the reduction in longest chain decrease the practical runtime of these dataflow algorithms.

Table 4.1: Critical path length (95% confidence intervals) and ratios of average critical path lengths across methods for different RMAT parameters. (Bolded numbers highlight critical path length ratios greater than 1.15.)

a	b	c	d	Uniform CI	Exponential CI	Linear CI	U/E	U/L	L/E
0.30	0.28	0.28	0.14	[2944; 2947]	[2623; 2625]	[2850; 2853]	1.123	1.033	1.087
0.40	0.24	0.24	0.12	[4950; 4953]	[4680; 4683]	[4859; 4862]	1.058	1.019	1.038
0.50	0.20	0.20	0.10	[6968; 6971]	[6643; 6647]	[6845; 6848]	1.049	1.018	1.030
0.60	0.16	0.16	0.08	[8353; 8357]	[7949; 7952]	[8175; 8178]	1.051	1.022	1.028
0.70	0.12	0.12	0.06	[9211; 9214]	[8738; 8740]	[8987; 8990]	1.054	1.025	1.029
0.30	0.28	0.17	0.25	[2111; 2113]	[2064; 2066]	[2103; 2105]	1.023	1.004	1.019
0.30	0.28	0.25	0.17	[2633; 2635]	[2437; 2439]	[2583; 2585]	1.080	1.019	1.060
0.30	0.28	0.34	0.08	[3782; 3785]	[3112; 3115]	[3510; 3513]	**1.215**	1.077	1.128
0.30	0.35	0.14	0.21	[2625; 2627]	[2047; 2049]	[2402; 2404]	**1.282**	1.093	**1.173**
0.30	0.35	0.21	0.14	[3064; 3067]	[2464; 2467]	[2825; 2827]	**1.243**	1.085	1.146
0.30	0.35	0.28	0.07	[3959; 3962]	[3221; 3225]	[3644; 3647]	**1.229**	1.086	1.131
0.30	0.42	0.11	0.17	[3632; 3635]	[2181; 2183]	[2880; 2883]	**1.665**	**1.261**	**1.321**
0.30	0.42	0.17	0.11	[3821; 3824]	[2433; 2436]	[3061; 3064]	**1.570**	**1.248**	**1.258**
0.30	0.42	0.22	0.06	[4343; 4346]	[3110; 3113]	[3685; 3688]	**1.396**	**1.178**	**1.185**
0.30	0.49	0.08	0.13	[4852; 4855]	[2245; 2249]	[3437; 3441]	**2.160**	**1.411**	**1.530**
0.30	0.49	0.13	0.08	[4775; 4778]	[2505; 2508]	[3325; 3330]	**1.906**	**1.435**	**1.328**
0.30	0.49	0.17	0.04	[5052; 5056]	[3151; 3155]	[3883; 3887]	**1.603**	**1.301**	**1.232**
0.40	0.24	0.14	0.22	[3246; 3249]	[3185; 3188]	[3242; 3245]	1.019	1.001	1.018
0.40	0.24	0.22	0.14	[4571; 4574]	[4377; 4380]	[4509; 4512]	1.044	1.014	1.030
0.40	0.24	0.29	0.07	[5946; 5950]	[5500; 5504]	[5759; 5763]	1.081	1.032	1.047
0.40	0.30	0.12	0.18	[4026; 4029]	[3457; 3460]	[3811; 3814]	**1.165**	1.056	1.102
0.40	0.30	0.18	0.12	[5003; 5006]	[4551; 4555]	[4821; 4825]	1.099	1.038	1.059
0.40	0.30	0.24	0.06	[6141; 6144]	[5652; 5656]	[5932; 5935]	1.086	1.035	1.049
0.40	0.36	0.10	0.14	[4903; 4906]	[3767; 3771]	[4333; 4336]	**1.301**	1.132	1.150
0.40	0.36	0.14	0.10	[5506; 5509]	[4637; 4641]	[5071; 5075]	**1.187**	1.086	1.094
0.40	0.36	0.19	0.05	[6383; 6387]	[5684; 5688]	[6045; 6049]	1.123	1.056	1.063
0.40	0.42	0.07	0.11	[5667; 5670]	[3901; 3906]	[4639; 4643]	**1.452**	**1.221**	**1.189**
0.40	0.42	0.11	0.07	[6196; 6199]	[4927; 4932]	[5478; 5483]	**1.257**	1.131	1.112
0.40	0.42	0.14	0.04	[6670; 6674]	[5681; 5685]	[6132; 6136]	**1.174**	1.088	1.079
0.50	0.20	0.12	0.18	[5009; 5012]	[4881; 4884]	[4981; 4984]	1.026	1.006	1.020
0.50	0.20	0.18	0.12	[6489; 6492]	[6213; 6216]	[6399; 6402]	1.044	1.014	1.030
0.50	0.20	0.24	0.06	[7813; 7816]	[7365; 7368]	[7616; 7620]	1.061	1.026	1.034
0.50	0.25	0.10	0.15	[5687; 5690]	[5230; 5234]	[5515; 5519]	1.087	1.031	1.054
0.50	0.25	0.15	0.10	[6920; 6924]	[6500; 6504]	[6748; 6751]	1.065	1.026	1.038
0.50	0.25	0.20	0.05	[7984; 7987]	[7503; 7507]	[7766; 7770]	1.064	1.028	1.035
0.50	0.30	0.08	0.12	[6294; 6297]	[5524; 5528]	[5943; 5946]	1.139	1.059	1.076
0.50	0.30	0.12	0.08	[7263; 7267]	[6644; 6648]	[6969; 6972]	1.093	1.042	1.049
0.50	0.30	0.16	0.04	[8073; 8076]	[7495; 7499]	[7786; 7789]	1.077	1.037	1.039
0.50	0.35	0.06	0.09	[6812; 6815]	[5778; 5781]	[6278; 6281]	**1.179**	1.085	1.087
0.50	0.35	0.09	0.06	[7537; 7540]	[6729; 6732]	[7108; 7111]	1.120	1.060	1.056
0.50	0.35	0.12	0.03	[8135; 8138]	[7420; 7423]	[7754; 7757]	1.096	1.049	1.045
0.60	0.16	0.10	0.14	[6491; 6495]	[6204; 6208]	[6406; 6409]	1.046	1.013	1.032
0.60	0.16	0.14	0.10	[7774; 7777]	[7418; 7422]	[7631; 7635]	1.048	1.019	1.029
0.60	0.16	0.19	0.05	[9077; 9080]	[8613; 8616]	[8843; 8846]	1.054	1.026	1.027
0.60	0.20	0.08	0.12	[6942; 6945]	[6427; 6431]	[6741; 6745]	1.080	1.030	1.049
0.60	0.20	0.12	0.08	[8257; 8260]	[7803; 7806]	[8044; 8048]	1.058	1.026	1.031
0.60	0.20	0.16	0.04	[9253; 9256]	[8754; 8757]	[8997; 9000]	1.057	1.028	1.028
0.60	0.24	0.06	0.10	[7261; 7264]	[6516; 6519]	[6926; 6929]	1.114	1.048	1.063
0.60	0.24	0.10	0.06	[8597; 8600]	[8041; 8044]	[8308; 8311]	1.069	1.035	1.033
0.60	0.24	0.13	0.03	[9283; 9286]	[8731; 8734]	[8987; 8990]	1.063	1.033	1.029
0.60	0.28	0.05	0.07	[7829; 7832]	[6958; 6962]	[7380; 7383]	1.125	1.061	1.061
0.60	0.28	0.07	0.05	[8537; 8540]	[7838; 7841]	[8157; 8160]	1.089	1.047	1.041
0.60	0.28	0.10	0.02	[9249; 9252]	[8631; 8634]	[8900; 8903]	1.072	1.039	1.031
0.70	0.12	0.07	0.11	[7229; 7232]	[6852; 6856]	[7101; 7105]	1.055	1.018	1.036
0.70	0.12	0.11	0.07	[8854; 8857]	[8424; 8427]	[8659; 8662]	1.051	1.023	1.028
0.70	0.12	0.14	0.04	[9813; 9816]	[9197; 9200]	[9514; 9517]	1.067	1.031	1.034
0.70	0.15	0.06	0.09	[7797; 7800]	[7252; 7255]	[7573; 7576]	1.075	1.030	1.044
0.70	0.15	0.09	0.06	[9093; 9096]	[8589; 8592]	[8850; 8853]	1.059	1.027	1.030
0.70	0.15	0.12	0.03	[10032; 10035]	[9347; 9350]	[9694; 9696]	1.073	1.035	1.037
0.70	0.18	0.05	0.07	[8267; 8270]	[7587; 7591]	[7941; 7945]	1.090	1.041	1.047
0.70	0.18	0.07	0.05	[9183; 9186]	[8599; 8602]	[8876; 8879]	1.068	1.035	1.032
0.70	0.18	0.10	0.02	[10074; 10076]	[9370; 9372]	[9700; 9702]	1.075	1.039	1.035
0.70	0.21	0.04	0.05	[8667; 8669]	[7899; 7902]	[8263; 8266]	1.097	1.049	1.046
0.70	0.21	0.05	0.04	[9168; 9171]	[8503; 8506]	[8798; 8800]	1.078	1.042	1.035
0.70	0.21	0.07	0.02	[9844; 9846]	[9198; 9200]	[9470; 9472]	1.070	1.039	1.030

Table 4.2: Graph basic statistics, critical path length (95% confidence intervals), and ratios of average critical path lengths across methods for several real world graphs. (Bolded numbers highlight critical path length ratios greater than 1.15.)

Name	Vertices	Edges	Max Degree	Clustering Coefficient	Diameter	Uniform CI	Exponential CI	Linear CI	U/E	U/L	L/E
CA-HepPh	89,209	118,521	491	0.6115	13	[1030; 1036]	[1040; 1045]	[1032; 1037]	0.991	0.999	0.992
Email-Enron	36,692	183,831	1,383	0.4970	11	[43437; 43720]	[38836; 38982]	[40688; 41002]	1.120	1.067	1.050
p2p-Gnutella04	10,879	39,994	103	0.0062	9	[911; 925]	[568; 575]	[728; 740]	**1.606**	**1.251**	**1.284**
roadNet-PA	1,090,920	1,541,898	9	0.0465	786	[49; 49]	[49; 50]	[48; 49]	0.990	1.010	0.980
soc-Epinions1	75,888	405,740	3,044	0.1378	14	[94793; 95270]	[88297; 88593]	[89488; 90034]	1.074	1.059	1.015
soc-pokec-relationships	1,632,804	22,301,964	14,854	0.1094	11	[118924; 119528]	[96958; 97239]	[100836; 101775]	**1.228**	**1.177**	1.043
web-Google	916,428	4,322,051	6,332	0.5143	21	[80466; 81618]	[18166; 18192]	[20577; 21084]	**4.458**	**3.891**	1.146
WikiTalk	2,394,385	4,659,565	100,029	0.0526	9	[1352414; 1357165]	[1101248; 1103043]	[1145942; 1151894]	**1.229**	**1.179**	1.042



(a) ca-HepPh (b) email-Enron

(c) p2p-Gnutella04 (d) roadNet-PA

(e) soc-Epinions1 (f) soc-pokec

(g) web-Google (h) wiki-Talk

Figure 4.4: Cumulative Density Functions of longest chain on real-world Graphs. All graphs (except ca-HepTh and roadNet-PA) show a major difference in critical path length across methods: `Exponential` and `Linear` have much shorter critical paths than `Uniform`.

CHAPTER 5: LONG PATHS IN ORIENTED $G_{N,P}$ GRAPHS

5.1 Introduction

After completing the heuristics experiment on RMAT graphs, we wanted a more precise explanation for the difference in performance of exponential and uniform. We also wanted to pursue an exact method for calculating the longest chain in these types of graphs. Since these methods were not immediately obvious to us, we searched through copious amounts of literature to see if this problem was already attempted by other researchers.

Almost all literature we found was specific to $G_{N,P}$ graphs – these results included bounds for chromatic numbers, counting Hamiltonian paths, characterizing subgraphs, and determining circuit complexity [44]. The most relevant result we found provided a lower bound for long paths in $G_{N,P}$ graphs. Krivelevich determined that if $p = \frac{c}{n}$ and $c \in O(\log n)$ is sufficiently large, then G has a path length at least $(1 - \frac{6 \log c}{c})$ [45]. Their proof method exploits properties specific to the $G_{N,P}$ model in order to apply a DFS algorithm on vertices with high degree. However, this result is limited to unweighted, undirected $G_{N,P}$ graphs and only provides a lower bound on the longest.

Other researchers have studied problem of edge-orientation from a slightly different perspective – Yen asks the question, "Given an undirected graph with constraints on edge weights, what is a directed graph that minimizes cost?" [46]. For our purposes, we do not need to find the orientation that yields the longest weighted path, but the longest path induced by orienting the edges from low to high ranking vertices. Hassin and Megiddo proposed an algorithm for finding an edge-orientation that does not increase the shortest-path between the members of a set of vertex pairs. Once again,

these interests differ from ours and finding these "ideal" orientations was proven to be NP-Complete when the input set of vertex pairs was greater than 2 [47].

We quickly discovered that our problem was largely unexplored and an existing solution did not exist, so we approached the problem of edge orientation for $G_{N,P}$ graphs. We looked to develop a framework that would allow us to pursue our original goal in the future. Therefore, in the remaining sections of this chapter, we provide a proof that probability of a long path in a $G_{N,P}$ graph goes to 0 as $N \to \infty$ if $\delta \in \log N$.

In Section 5.2, we bound the probability of the existence for an individual path by a function F. We also bound the existence of any long path by a function $G(N)$ which is expressed using F. In Section 5.3, we show that $G(N)$ goes to 0 when N approaches ∞, demonstrating that long paths have a low probability of existence. In Section 5.4, we provide experimental results that reinforce the proof techniques used in the previous sections. Furthermore, we share some concluding remarks to this chapter in Section 5.5.

5.2 Long Paths are Bounded by $G(N)$

Theorem 5.2.1. *In $G_{N,P}$ graphs with a uniformly random total order, as long as $P = \frac{c_1 \log N}{N}$, w.h.p there is no path L, such that $L \in \omega(\log N)$ for sufficiently large N.*

Definition 5.2.2. *$E(L,N,P)$ is an event, such that G contains a path of size exactly L.*

Definition 5.2.3. *$E_i(L,N,P)$ is an event that contains the sub-paths $[(v_i,1)$ AND $(1,x)$ of size $L-1]$ OR $[(v_i,2)$ AND $(2,x)$ of size $L-1]$ OR $[(v_i,3)$ AND $(3,x)$ of size $L-1]$ OR ...*

Remark 5.2.4. *The probability of having a path of size L is the probability of the event E for large L. We need to show that the sum over all L goes to 0 as N goes to ∞. This is a slightly stronger condition than $\mathbb{P}(E_0(L,N,P))$ goes to 0 as N approaches*

∞.

Lemma 5.2.5. $\mathbb{P}(E(L,N,P)) \leq \sum_{i=1}^{N} \mathbb{P}(E_0(L,N-i,P))$

Proof. We have $E(L,N,P) = \bigcup_{i=0}^{N} E_i(L,N,P)$ by construction of E. By countable subadditivity $\mathbb{P}(E(L,N,P)) \leq \sum_{i=0}^{N} \mathbb{P}(E_i(L,N,P))$.

We now show $\mathbb{P}(E_i(L,N,P)) = \mathbb{P}(E_0(L,N-i,P))$. Recall, edges only go from low index to high index, so no vertices with index between 0 and $i-1$ are part of the path in the $E_i(L,N,P)$ event. Since all edges are equiprobable, simply remove v_0 from the graph and relabel the vertex with the lowest remaining index to v_0. Clearly, we have a graph that preserves our path of size L with $N-i$ vertices. □

We have $\mathbb{P}(E_0(L,N,P) \leq \sum_{i=1}^{N} \mathbb{P}(Edge(0,i))\mathbb{P}(E_i(L-1,N,P)$ by definition of E_0 and countable subadditivity. Since $\mathbb{P}(E_i(L-1,N,P)) = \mathbb{P}(E_0(L-1,N-i,P))$, we have $\mathbb{P}(E_0(L,N,P) \leq \sum_{i=1}^{N} P * \mathbb{P}(E_0(L-1,N-i,P))$. We now define a function to analyze this expression.

Definition 5.2.6. *F(L,N,P)* $= \sum_{i=1}^{N} P * F(L-1,N-i,P)$ *with base case* $F(L > 0, N = 0, P) = 0$ *and* $F(L = 1, N > 0, P) = 1$.

Proposition 5.2.7. $\mathbb{P}(E_0(L,N,P)) \leq F(L,N,P)$

Definition 5.2.8. *G(N)* $= \sum_{L > c_2 \log N} \sum_{M \leq N} F(L, M, \frac{c_1 \log N}{N})$

Lemma 5.2.9. *If* $G(N)$ *goes to* 0 *as* N *approaches* ∞, *then the probability of a long path event also goes to* 0.

Proof.

$$LongPathEvent = \bigcup_{L > c_2 \log N} E(L,N,P) \tag{5.1}$$

$$= \bigcup_{L > c_2 \log N} \bigcup_{i=0}^{N-1} E_i(L,N,P) \tag{5.2}$$

$$\mathbb{P}(LongPathEvent) \leq \sum_{L > c_2 \log N} \sum_{M \leq N} F(L,M,P) \tag{5.3}$$

□

Lemma 5.2.10. $G(N) = \sum_{L > c_2 \log N} t(L, N)$, *such that* $t(L, N) = \left(\frac{c_1 \log N}{N}\right)^{L-1} \binom{N+1}{L}$

Proof. Let $H(L, N) = \sum_{i=1}^{N} H(L-1, N-i)$, such that $H(L > 0, N = 0) = 0$ and $H(L = 1, N > 0) = 1$. We will prove that $H(L, N) = \binom{N}{L-1}$ by induction on L. The base case is trivial: $H(L = 1, N > 0) = 1 = \binom{N}{0}$. We now assume that $H(L, N) = \binom{N}{L-1}$. We will now show $H(L+1, N) = \binom{N}{L}$ to complete the proof by induction on L.

$$H(L+1, N) = \sum_{i=1}^{N} H(L, N-i) = \sum_{i=1}^{N} \binom{N-i}{L-1} = \sum_{i=L-1}^{N-1} \binom{i}{L-1} = \binom{N}{L} \tag{5.4}$$

Let $v = [v_0, ..., v_{N-1}], v_i \in \{0, 1\}$ be a vector that encodes the vertices taken by a path, such that $v_i = 1$ if the path uses $N = v_i$. Clearly, v has $(L-1)$ $1s$ and $(N-L+1)$ $0s$. Hence, $\binom{N}{L-1}$ counts how many vectors have exactly $(L-1)$ $1s$.

$$F(L, N, P) = \sum_{i=1}^{N} P * F(L-1, N-i, P) \tag{5.5}$$

$$= P^{L-1} \sum_{i=1}^{N} H(L-1, N-i) \tag{5.6}$$

$$= P^{L-1} H(L, N) \tag{5.7}$$

$$= P^{L-1} \binom{N}{L-1} \tag{5.8}$$

Therefore,

$$G(N) = \sum_{L > c_2 \log N} \sum_{M \leq N} P^{L-1} \binom{N}{L-1} = \sum_{L > c_2 \log N} P^{L-1} \binom{N+1}{L} = \sum_{L > c_2 \log N} t(L, N) \tag{5.9}$$

□

5.3 Analysis of G(N)

Lemma 5.3.1. *G(N) is dominated by first term, such that* $G(N) \leq Bt(c_2 \lg(N), N)$.

Proof. We need to show that $t(L+1, N) \leq \frac{1}{c} t(L, N)$ for $L > c_2 \log N$ and $c > 1$, which implies $\frac{1}{1-\frac{1}{c}}$ is constant. For $L > \frac{N}{2}$ both terms decrease, so we only care about $c_2 \log N < L < \frac{N}{2}$.

$$t(L+1, N) = \left(\frac{c_1 \log N}{N}\right)^L \binom{N+1}{L+1} \tag{5.10}$$

$$= \left(\frac{c_1 \log N}{N}\right)^{L-1} \left(\frac{c_1 \log N}{N}\right) \left(\frac{N-(L+1)+1}{L+1}\right) \binom{N+1}{L} \tag{5.11}$$

$$= \left(\frac{c_1 \log N}{N}\right) \left(\frac{N-L}{L+1}\right) t(L, N) \tag{5.12}$$

If $(\frac{c_1 \log N}{N})(\frac{N-L}{L+1}) < B < 1$, then $(\frac{c_1 \log N}{N})(N-L) < B(L+1)$. If $B(L+1) > c_1 \log N$, then $(c_1 \log N) - (\frac{c_1 \log N(L+1)}{N}) < B(L+1)$. This is sufficient. Hence, $B > \frac{c_1 \log N}{L+1}$. We have $L > c_2 \lg(N)$, so $B > \frac{c_1 \log N}{c_2 \log N+1}$. For $B = \frac{c_1}{c_2}$, the lemma is true if $c_2 > c_1$. □

Lemma 5.3.2. $\lim_{N\to\infty} t'(N) = 0$.

Proof. Recall that $t(L, N) = \frac{c_1 \log_2(N)}{N}^{L-1} \binom{N+1}{L}$. Hence, $t'(N) = \frac{c_1 \log_2(N)}{N}^{c_2 \log_2(N)-1} \binom{N+1}{c_2 \log_2(N)}$. It is sufficient to show $\lim_{N\to\infty} \frac{t'(N)}{t'(2N)} \geq \mathit{constant} > 1$.

$$\frac{t'(N)}{t'(2N)} = \frac{\frac{c_1 \log_2(N)}{N}^{c_2 \log_2(N)-1} \binom{N+1}{c_2 \log_2(N)}}{\frac{c_1 \log_2(N)+c_1}{2N}^{c_2 \log_2(N)-1+c_2} \binom{2N+1}{c_2 \log_2(N)+c_2}} \tag{5.13}$$

$$= \frac{(2N)^{c_2 \log_2(N)-1+c_2}}{N^{c_2 \log_2(N)-1}} \frac{(c_1 \log_2(N))^{c_2 \log_2(N)-1}}{(c_1 \log_2(N)+c_1)^{c_2 \log_2(N)-1+c_2}} \frac{\binom{N+1}{c_2 \log_2(N)}}{\binom{2N+1}{c_2 \log_2(N)+c_2}} \tag{5.14}$$

$$= 2^{c_2 \log_2(N)-1}(2N)^{c_2} \left(\frac{c_1 \log_2(N)}{c_1 \log_2(N)+c_1}\right)^{c_2 \log_2(N)-1} \left(\frac{1}{c_1 \log_2(N)+c_1}\right)^{c_2} \frac{\binom{N+1}{c_2 \log_2(N)}}{\binom{2N+1}{c_2 \log_2(N)+c_2}} \tag{5.15}$$

$$= (2N)^{c_2} \left(\frac{1}{c_1 \log_2(N)+c_1}\right)^{c_2} \left(\frac{\log_2(N)}{\log_2(N)+1}\right)^{c_2 \log_2(N)-1} 2^{c_2 \log_2(N)-1} \frac{\binom{N+1}{c_2 \log_2(N)}}{\binom{2N+1}{c_2 \log_2(N)+c_2}} \tag{5.16}$$

$$= \left(\frac{2N}{c_1 \log_2(N)+c_1}\right)^{c_2} \left(\frac{2 \log_2(N)}{\log_2(N)+1}\right)^{c_2 \log_2(N)-1} \frac{\binom{N+1}{c_2 \log_2(N)}}{\binom{2N+1}{c_2 \log_2(N)+c_2}} \tag{5.17}$$

For $k \in o(n)$, we can use the following approximation derived from Sterling's formula: $\binom{n}{k} \sim \left(\frac{ne}{k}\right)^k (2\pi k)^{-1/2} e^{\left(-\frac{k^2}{2n}(1+o(1))\right)}$. We first study the ratio $\frac{\binom{N+1}{c_2 \log_2(N)}}{\binom{2N+1}{c_2 \log_2(N)+c_2}}$ in the light of that approximation.

$$\frac{\binom{N+1}{c_2\log_2(N)}}{\binom{2N+1}{c_2\log_2(N)+c_2}} = \frac{\left(\frac{(N+1)e}{c_2\log_2(N)}\right)^{c_2\log_2(N)} (2\pi(c_2\log_2(N))^{-\frac{1}{2}} e^{\left(-\frac{(c_2\log_2(N))^2}{2(N+1)}(1+o(1))\right)}}{\left(\frac{(2N+1)e}{c_2\log_2(N)+c_2}\right)^{c_2\log_2(N)+c_2} (2\pi(c_2\log_2(N)+c_2))^{-\frac{1}{2}} e^{\left(-\frac{(c_2\log_2(N)+c_2)^2}{2(2N+1)}(1+o(1))\right)}} \tag{5.18}$$

$$\begin{aligned} &= \frac{e^{c_2\log_2 N}}{e^{c_2\log_2 N+c_2}} \frac{e^{\left(-\frac{(c_2\log_2(N))^2}{2(N+1)}(1+o(1))\right)}}{e^{\left(-\frac{(c_2\log_2(N)+c_2)^2}{2(2N+1)}(1+o(1))\right)}} \left(\frac{2\pi(c_2\log_2(N)+c_2)}{2\pi(c_2\log_2(N))}\right)^{\frac{1}{2}} \\ &\times \frac{(N+1)^{c_2\log_2(N)}}{(2N+1)^{c_2\log_2(N)+c_2}} \frac{(c_2\log_2(N)+c_2)^{c_2\log_2(N)+c_2}}{(c_2\log_2(N))^{c_2\log_2(N)}} \end{aligned} \tag{5.19}$$

$$\begin{aligned} &= \left(\frac{1}{e^{c_2}}\right)\left(e^{\frac{(c_2\log_2(N)+c_2)^2}{2N+1}-\frac{(c_2\log_2(N))^2}{N+1}}\right) \frac{(N+1)^{c_2\log_2(N)}}{(2N+1)^{c_2\log_2(N)+c_2}} \\ &\times \frac{(c_2\log_2(N)+c_2)^{c_2\log_2(N)+c_2+\frac{1}{2}}}{(c_2\log_2(N))^{c_2\log_2(N)+\frac{1}{2}}} \end{aligned} \tag{5.20}$$

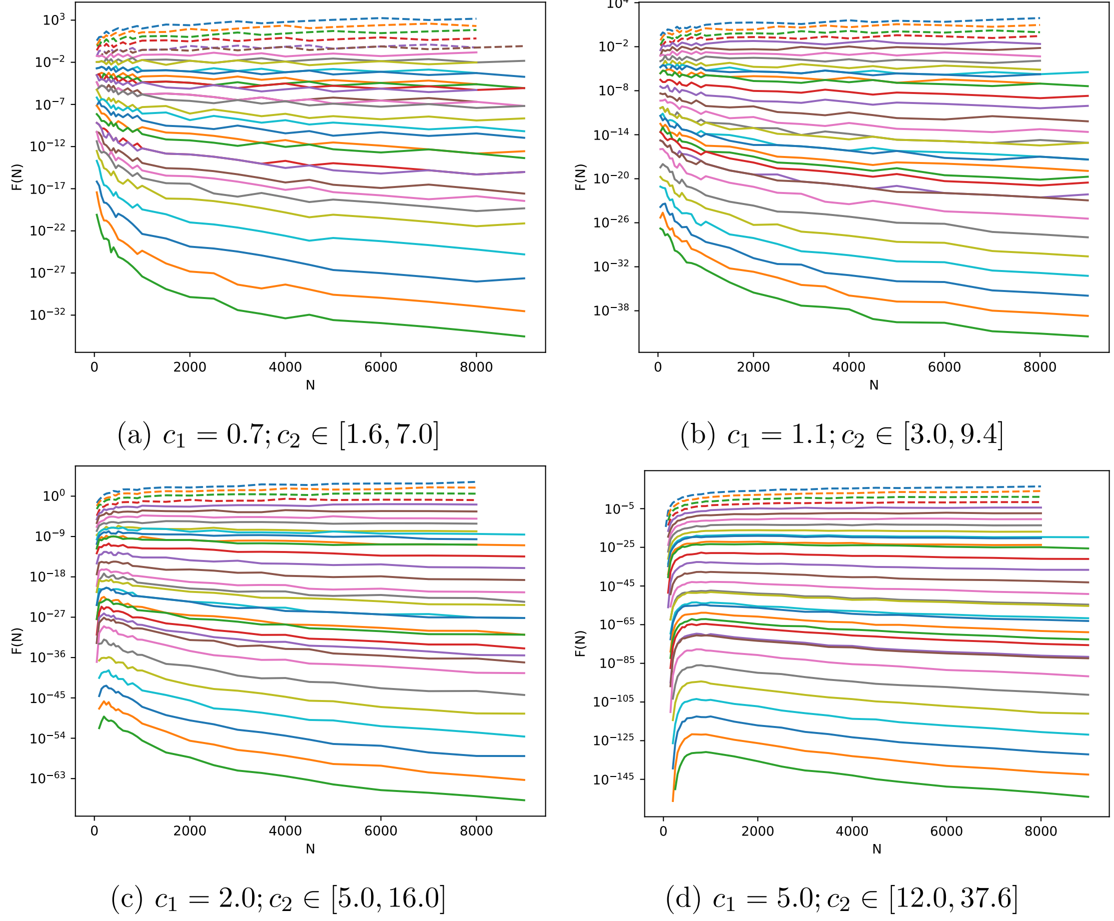
$$\begin{aligned} &= \left(\frac{1}{e^{c_2}}\right)\left(e^{\frac{(c_2\log_2(N)+c_2)^2}{2N+1}-\frac{(c_2\log_2(N))^2}{N+1}}\right) \left(\frac{c_2\log_2(N)+c_2}{2N+1}\right)^{c_2} \\ &\times \left(\frac{N+1}{2N+1}\right)^{c_2\log_2(N)} \left(\frac{\log_2(N)+1}{\log_2(N)}\right)^{c_2\log_2(N)+\frac{1}{2}} \end{aligned} \tag{5.21}$$

Hence we have,

$$\begin{aligned} \frac{t'(N)}{t'(2N)} &= \left(\frac{2N}{2N+1}\right)^{c_2} \left(\frac{c_2(\log_2(N)+1)}{c_1(\log_2(N)+1)}\right)^{c_2} \left(\frac{2(N+1)\log_2(N)(\log_2(N)+1)}{(2N+1)\log_2(N)(\log_2(N)+1)}\right)^{c_2\log_2(N)} \\ &\times \left(\frac{\log_2(N)+1}{2\log_2(N)}\right)\left(\frac{\log_2(N)+1}{\log_2(N)}\right)^{\frac{1}{2}} \left(\frac{1}{e^{c_2}}\right)\left(e^{\frac{(c_2\log_2(N)+c_2)^2}{2N+1}-\frac{(c_2\log_2(N))^2}{N+1}}\right) \end{aligned} \tag{5.22}$$

$$\begin{aligned} &= \left(\frac{2N}{2N+1}\right)^{c_2}\left(\frac{c_2}{c_1}\right)^{c_2}\left(\frac{2(N+1)}{2N+1}\right)^{c_2\log_2(N)}\left(\frac{1}{2}\right)\left(\frac{\log_2(N)+1}{\log_2(N)}\right)^{\frac{3}{2}} \\ &\times \left(\frac{1}{e^{c_2}}\right)\left(e^{\frac{(c_2\log_2(N)+c_2)^2}{2N+1}-\frac{(c_2\log_2(N))^2}{N+1}}\right) \end{aligned} \tag{5.23}$$

$$\begin{aligned} &= \left(\frac{1}{2}\right)\left(\frac{c_2}{e\cdot c_1}\right)^{c_2}\left(\frac{2N}{2N+1}\right)^{c_2}\left(\frac{2N+2}{2N+1}\right)^{c_2\log_2(N)} \\ &\times \left(\frac{\log_2(N)+1}{\log_2(N)}\right)^{\frac{3}{2}}\left(e^{\frac{(c_2\log_2(N)+c_2)^2}{2N+1}-\frac{(c_2\log_2(N))^2}{N+1}}\right) \end{aligned} \tag{5.24}$$



(a) $c_1 = 0.7; c_2 \in [1.6, 7.0]$

(b) $c_1 = 1.1; c_2 \in [3.0, 9.4]$

(c) $c_1 = 2.0; c_2 \in [5.0, 16.0]$

(d) $c_1 = 5.0; c_2 \in [12.0, 37.6]$

Figure 5.1: Experimental Results for Long Paths in $G_{N,P}$

Therefore,

$$\lim_{N\to\infty} \frac{t'(N)}{t'(2N)} = \left(\frac{1}{2}\right) \cdot \left(\frac{c_2}{e \cdot c_1}\right)^{c_2} \cdot 1 \cdot 1 \cdot 1 \cdot 1 = \left(\frac{1}{2}\right) \left(\frac{c_2}{e \cdot c_1}\right)^{c_2} \tag{5.25}$$

Any choice of c_1, c_2, such that $\left(\frac{c_2}{e \cdot c_1}\right)^{c_2} > 2$ yields a constant limit for $\frac{t'(N)}{t'(2N)}$ greater than 1.

$\square$

5.4 Experimental Results

We gathered experimental results to further validate the methods we developed in previous sections. We analyzed the behavior of the function $F(L, N, P)$ by sampling

the c_1, c_2 parameter space. Figure 5.1 includes a selection of the parameters used in our experiment. Each subfigure uses a different c_1 and each line represents a different c_2. Each line shows the value of F at each N in the range with the associated values for c_1, c_2.

It is important to recall a few important details from the proof to ensure that the figures do not appear odd at first glance. We prove that F goes to 0 as N becomes sufficiently large; however, this does not mean that the function is strictly decreasing. In Figure 5.1a on the bottom line $c_2 = 7.0$, F is increasing on the interval from $(4000, 4500)$, yet F is already approaching 0 for larger N.

The behavior of F appears to be determined by the value of $\left(\frac{c_2}{e \cdot c_1}\right)^{c_2}$ from Equation 5.25. If $\left(\frac{c_2}{e \cdot c_1}\right)^{c_2} > 2$, we use a solid line to illustrate that F should be approaching 0 as N gets sufficiently large. If $\left(\frac{c_2}{e \cdot c_1}\right)^{c_2} \leq 2$, then F may not be approaching 0 by the largest N run by our experiment, so we use a dotted line to illustrate F. F usually starts approaching 0 more quickly when $\left(\frac{c_2}{e \cdot c_1}\right)^{c_2}$ is much larger than 2.

Lemma 5.2.9 dictates that F is a bound on the probability of a long path event not the probability of the event itself, which is why some values of F may be greater than 1. F may also be increasing for small n because we only require F to be decreasing after $c_2 \log N$. We use an approximation for $\binom{n}{k}$ in Lemma 5.3.2 that is only valid when k is roughly in $o(n)$. Hence, our result may not hold when n is small because $c_2 \log n$ could be almost as large as N. Similarly, we assume that $\delta \in O(\log n)$, so if c_1 is large and n is small, then $c_1 \log n$ can potentially be as large as n.

This experiment required a substantial amount of computational resources, so jobs were completed using the cluster at the University of North Carolina at Charlotte. Even so, for large values of N, the experiment was likely to time out or throw an out of memory exception. Therefore, we do not have any data for when $N > 10,000$.

5.5 Conclusion

In this chapter, we proved that there are no long paths in $G_{N,P}$ graphs after N is sufficiently large. We also presented experimental results as an additional basis for our claims. These experiments helped develop our intuition for this problem while searching for an exact solution.

We looked exclusively at $G_{N,P}$ graphs; however, it is possible to use other graph models. This extension is non-trivial because several methods discussed in this chapter take advantage of properties unique to $G_{N,P}$ graphs.

In Lemma 5.2.5, we relabel each vertex based on the total order of the graph. Each edge in a $G_{N,P}$ is equiprobable, so we are free to reorder each vertex without changing the result. This is not true for RMAT graphs because the probability of an edge between each pair of vertices depends on their order. Our result is only valid for uniform edge orientations for a similar reason. Recall that the `Exponential` method from Section 4.3 is an ordering, such that each vertex v draws a random number in $[0; 2^{\delta(v)})$. Since vertices are ordered as a function of degree, we are unable to reorder them without changing the total order.

In future work, we would like to address these limitations by extending our argument to support additional graph models and edge orientations.

CHAPTER 6: CONCLUSION

6.1 Summary of Results

We started this dissertation by approaching the problem of interval coloring for stencil graphs. We were interested in stencil graphs because they can be used to represent load balancing spatial applications in parallel computing. We solved subproblems on cliques, bipartite graphs, and odd cycles. We proved interval coloring of 3D 27-pt stencil is NP-Complete. We designed heuristics, including 2-approximation for 2D 9-pt stencil, 4-approximation for 3D 27-pt stencil, greedy with post optimization. We evaluated these heuristics and confirmed model validity on a real-world application.

We became interested in dataflow algorithms as a result of our work on interval coloring. We provided a model for dataflow algorithms as a distributed graph coloring problem. We presented new ways to generate partial orderings and provided a theoretical argument for why they are sound. We studied the behavior of algorithms using different partial orders on both randomly generated RMAT graphs and graphs from real-world applications. We provided an argument using statistical evidence to show that our methods perform usually better.

We searched for a formal proof to explain the performance discrepancy of the different methods used in the RMAT and real-world experiment. We discovered that the construction of RMAT graphs was inherently problematic to existing methods of random graph analysis. Hence, we looked to $G_{N,P}$ graphs for inspiration. We were able to show that given modest conditions on edge probability, the likelihood of a long path goes to 0 once N becomes sufficiently large.

6.2 Open Questions

We are interested in deploying our interval coloring solution on a graph processing software but that work remains to be completed. We do not know if there are other types of applications not modeled by stencil graphs that can benefit from interval coloring. Additionally, we would like to know if there are any other types of graphs that have an exploitable structure similar to grids.

We looked at dataflow algorithms from the perspective of graph coloring; however, there are other types of applications such as bipartite matching. We would be interested in guaranteed quality of matching that has a good critical path runtime. We looked at the deployment of Luby's Algorithm in particular, but it should be possible to extend our application to include Jones-Plassman. Furthermore, we would like to know if there are there multi-stage properties that would enhance the accuracy of our result. The primary advantage of dataflow algorithms is that they can be processed massively in parallel. Adding sequential operations would necessarily increase runtime, but it may be advantageous for solution quality.

We only consider unweighted paths in our work on random graphs, so we would like to pursue the problem for weighted paths. We would like to support for different graph models and edge orientations as well.

REFERENCES

[1] R. Dennard, F. Gaensslen, H.-N. Yu, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[2] Y. Robert, *Task Graph Scheduling*, pp. 2013–2025. Boston, MA: Springer US, 2011.

[3] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, p. 406â471, dec 1999.

[4] W. Wang and L. Ying, "Resource allocation for data-parallel computing in networks with data locality," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 933–939, 2016.

[5] B. A and A. X. M, "A simple model to optimize general flow-shop scheduling problems with known break down time and weights of jobs," *Procedia Engineering*, vol. 38, pp. 191–196, 2012. INTERNATIONAL CONFERENCE ON MODELLING OPTIMIZATION AND COMPUTING.

[6] T. Gonzalez and S. Sahni, "Open shop scheduling to minimize finish time," *J. ACM*, vol. 23, p. 665â679, oct 1976.

[7] L. KuÄera, "Parallel computation and conflicts in memory access," *Information Processing Letters*, vol. 14, no. 2, pp. 93–96, 1982.

[8] P. A. Golovach, M. Johnson, D. Paulusma, and J. Song, "A survey on the computational complexity of colouring graphs with forbidden subgraphs," *CoRR*, vol. abs/1407.1482, 2014.

[9] D. Durrman and E. Saule, "Coloring the vertices of 9-pt and 27-pt stencils with intervals," in *Proc. Of IPDPS*, May 2022.

[10] D. Durrman and E. Saule, "Optimizing the critical path of distributed dataflow graph algorithms," in *Proc. Of IPDPSW; PDCO*, May 2023.

[11] E. Saule, D. Panchananam, A. Hohl, W. Tang, and E. Delmelle, "Parallel space-time kernel density estimation," in *Proceedings of ICPP 2017*, 2017.

[12] D. Kratsch, "Finding the minimum bandwidth of an interval graph," *Information and Computation*, vol. 74, no. 2, pp. 140–158, 1987.

[13] M. Bouchard, M. ÄangaloviÄ, and A. Hertz, "On a reduction of the interval coloring problem to a series of bandwidth coloring problems," *Journal of Scheduling*, vol. 13, pp. 583–595, 12 2010.

[14] Z. Shao, Z. Li, B. Wang, S. Wang, and X. Zhang, "Interval edge-coloring: A model of curriculum scheduling," *AKCE International Journal of Graphs and Combinatorics*, vol. 17, no. 3, pp. 725–729, 2020.

[15] M. Cangalovic and J. A. M. Schreuder, "Exact colouring algorithm for weighted graphs applied to timetabling problems with lectures of different lengths," *European Journal of Operational Research*, vol. 51, no. 2, pp. 248–258, 1991.

[16] M. R. Garey and D. S. Johnson, *Computers and Intractability.* Freeman, San Francisco, 1979.

[17] D. de Werra and A. Hertz, "Consecutive colorings of graphs," *Zeitschrift für Operations Research*, vol. 32, pp. 1–8, Jan 1988.

[18] D. W. Matula, "A min-max theorem for graphs with application to graph coloring," *SIAM Review*, vol. 10, pp. 481–482, 1968.

[19] A. H. Gebremedhin, F. Manne, and A. Pothen, "What color is your jacobian? Graph coloring for computing derivatives," *SIAM Review*, vol. 47, no. 4, pp. 629–705, 2005.

[20] D. J. A. Welsh and M. B. Powell, "An upper bound for the chromatic number of a graph and its application to timetabling problems," *The Computer Journal*, vol. 10, pp. 85–86, 1967.

[21] D. W. Matula and L. L. Beck, "Smallest-last ordering and clustering and graph coloring algorithms," *Jounal of the ACM*, vol. 30, pp. 417–427, July 1983.

[22] J. C. Culberson, "Iterated greedy graph coloring and the difficulty landscape," Tech. Rep. TR 92-07, University of Alberta, June 1992.

[23] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, pp. 406–471, Dec. 1999.

[24] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM Journal on Applied Mathematics*, vol. 17, pp. 416–429, Mar. 1969.

[25] D. B. West, *Introduction to Graph Theory.* Prentice-Hall, 1996.

[26] Y. Asahiro, J. Jansson, E. Miyano, and et al., "Approximation algorithms for the graph orientation minimizing the maximum weighted outdegree," *J Comb Optim*, vol. 22, pp. 78–96, 2011.

[27] R. Cole and U. Vishkin, "Deterministic coin tossing with applications to optimal parallel list ranking," *Information and Control*, vol. 70, no. 1, pp. 32–53, 1986.

[28] K. Appel and W. Haken, "Every planar map is four-colorable, ii: Reducibility," *Illinois J. Math.*, no. 21, pp. 491–567, 1977.

[29] D. Zuckerman, “Linear degree extractors and the inapproximability of max clique and chromatic number,” *Theory of Computing*, vol. 3, pp. 103–128, 2007.

[30] D. Brélaz, “New methods to color the vertices of a graph,” *Commun. ACM*, vol. 22, pp. 251–256, April 1979.

[31] D. Orden, J. M. Gimenez-Guzman, I. Marsa-Maestre, and E. De la Hoz, “Spectrum graph coloring and applications to wi-fi channel assignment,” *Symmetry*, vol. 10, no. 3, 2018.

[32] G. E. Blelloch, J. T. Fineman, and J. Shun, “Greedy sequential maximal independent set and matching are parallel on average,” in *Proceedings of the Twenty-Fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA ’12, (New York, NY, USA), p. 308â317, Association for Computing Machinery, 2012.

[33] D. Nicol, “Rectilinear partitioning of irregular data parallel computations,” *Journal of Parallel and Distributed Computing*, vol. 23, pp. 119–134, 1994.

[34] C. W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” *SIGGRAPH Computer Graphics*, vol. 21, p. 25â34, Aug. 1987.

[35] B. M. E. Moret, “Planar NAE3SAT is in P,” *SIGACT News*, vol. 19, p. 51â54, June 1988.

[36] U. V. Çatalyürek, J. Feo, A. H. Gebremedhin, M. Halappanavar, and A. Pothen, “Graph coloring algorithms for multi-core and massively multithreaded architectures,” *Parallel Comput.*, vol. 38, p. 576â594, oct 2012.

[37] A. E. Sarıyüce, E. Saule, and U. V. Catalyurek, “Scalable hybrid implementation of graph coloring using MPI and OpenMP,” in *26th International Symposium on Parallel and Distributed Processing, Workshops and PhD Forum (IPDPSW), Workshop on Parallel Computing and Optimization (PCO)*, May 2012.

[38] M. Luby, “A simple parallel algorithm for the maximal independent set problem,” *SIAM Journal on Computing*, vol. 15, no. 4, pp. 1036–1053, 1986.

[39] M. T. Jones and P. E. Plassmann, “A parallel graph coloring heuristic,” *SIAM Journal on Scientific Computing*, vol. 14, no. 3, pp. 654–669, 1993.

[40] R. R. McCune, T. Weninger, and G. Madey, “Thinking like a vertex: A survey of vertex-centric frameworks for large-scale distributed graph processing,” *ACM Comput. Surv.*, vol. 48, oct 2015.

[41] A. E. Sarıyüce, E. Saule, and U. V. Catalyurek, “Improving graph coloring on distributed memory parallel computers,” in *18th Annual International Conference on High Performance Computing*, 2011.

[42] C. F. Deepayan Chakrabarti, Yiping Zhany, "R-mat: A recursive model for graph mining," in *Proc. of SIAM International Conference on Data Mining (SDM)*, 2004.

[43] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: an approach to modeling networks," *Journal of Machine Learning Research*, vol. 11, pp. 985–1042, 2010.

[44] N. Alon and J. Spencer, *The Probabilistic Method.* Wiley Series in Discrete Mathematics and Optimization, Wiley, 2015.

[45] A. Frieze and M. Karoński, *Introduction to Random Graphs.* Introduction to Random Graphs, Cambridge University Press, 2016.

[46] W. C.-K. Yen, "The edge-orientation problem and some of its variants on weighted graphs," *Information Sciences*, vol. 176, no. 19, pp. 2791–2816, 2006.

[47] R. Hassin and N. Megiddo, "On orientations and shortest paths," *Linear Algebra and its Applications*, vol. 114-115, pp. 589–602, 1989. Special Issue Dedicated to Alan J. Hoffman.