Coloring the Vertices of 9-pt and 27-pt Stencils with Intervals

Dante Durrman
Dept. Mathematics
UNC Charlotte
Charlotte, NC
ddurrman@uncc.edu

Erik Saule
Dept. Computer Science
UNC Charlotte
Charlotte, NC
esaule@uncc.edu

***Abstract*—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.**

In this paper, we are interested in studying this problem for particular classes of graphs, namely stencil graphs. Stencil graphs appear naturally in the parallelisation 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.

***Index Terms*—interval vertex coloring, stencils, np completeness, approximation algorithms, heuristics**

I. 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 [1].

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 paper, 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 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 [2] 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 [3], or visualization of spatio-temporal data [4].

In this paper, 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 II. We study special cases in Section III 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

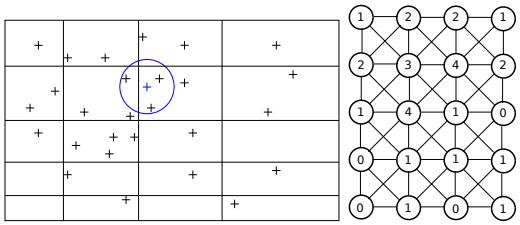


Fig. 1: Application leading to a 5×4 9-pt stencil graph

in Section IV that the problem of interval coloring of 27-pt 3D stencil with a small number of colors is NP-Complete. Section V 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 VI. In Section VII, we integrate our heuristics in a Space-Time Kernel Density Estimation application [4] and show that the number of colors derived by the heuristics correlates with the runtime of the application.

II. Interval Coloring Problem of Stencils

A. Problem Definition

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

Definition 1 (Interval Vertex Coloring (IVC))**.** *Let $G = (V, E)$ be an undirected graph and $w : V \to \mathbb{Z}^+$ 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 $start : V \to \mathbb{Z}^+$. 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 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 (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).

B. Related Works

The interval coloring problem has a long-established history with multiple variants and classical applications. Bandwidth problems [5], [6], scheduling problems [7], and timetabling problems [8] are just a few applications.

Since the complexity of the interval coloring problem for general graphs is known to be NP-Hard [1], several authors have provided bounds on the generalized chromatic number [9]. 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 [10]. 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 [11]. Popular orderings are Largest First [12], and Smallest Last [13]. Some post optimization techniques have proven to be particularly effective, such as recoloring [14].

III. 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.

A. 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 ($\forall 0 \leq i < X, 0 \leq j < Y, 0 \leq k < Z, maxcolor^* \geq w(i,j,k) + w(i,j+1,k) + w(i+1,j,k) + w(i+1,j+1,k) + w(i,j,k+1) + w(i,j+1,k+1) + w(i+1,j,k+1) + w(i+1,j+1,k+1)$).

B. 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.

C. 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 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 4 (maxpair)**.** *Let $maxpair$ be the maximum sum of any 2 consecutive terms: $maxpair = \max_i w(i, i+1)$*

$$\begin{bmatrix} 0 & 18 & 6 & 0 \\ 7 & 0 & 0 & 18 \\ 18 & 0 & 6 & 0 \\ 0 & 7 & 0 & 0 \end{bmatrix}$$

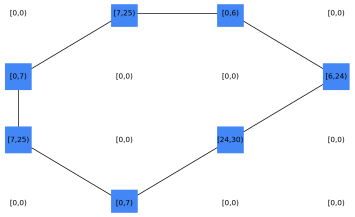


Fig. 2: Odd Cycle Instance and its Optimal Coloring

Definition 5 (minchain3)**.** *Let $minchain3$ be the minimum sum of any 3 consecutive terms: $minchain3 = \min_i w(i, i+1, i+2)$*

Theorem 1. *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 2. *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. □

Lemma 3. *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. □

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.

D. 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) an instance whose optimal coloring uses stricly more color than either lower bounds.

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.)

IV. 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 [15]. 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.

Fig. 3: Optimal Coloring of 2 Neighboring Cycles

Lemma 4. *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. □

Lemma 5. *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 $\mathbb{Z}^3$. 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: $\forall(x \leq n, z \leq 2m)$,

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

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

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 $z = 2j + 1$):

$$W_{2j+1} = \begin{bmatrix} 0 & 7 & 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7 & 0 & 0 & 7 & 0 & 0 & 0 & 7 & 7 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 & 3 & 0 & 0 & 7 \\ 7 & 7 & 0 & 0 & 7 & 3 & 3 & 0 & 0 & 7 \\ 0 & 0 & 7 & 0 & 0 & 0 & 0 & 7 & 0 & 7 \\ 7 & 0 & 0 & 7 & 0 & 0 & 7 & 0 & 0 & 7 \\ 0 & 7 & 0 & 0 & 7 & 7 & 0 & 0 & 0 & 7 \\ 0 & 0 & 7 & 0 & 0 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 7 & 7 & 7 & 7 & 7 & 0 & 0 \end{bmatrix} \quad (1)$$

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 6. *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.

V. Heuristics

A. 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 7. *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)*.

B. 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 III-B 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 8. *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 9. *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)*.

VI. Experiments

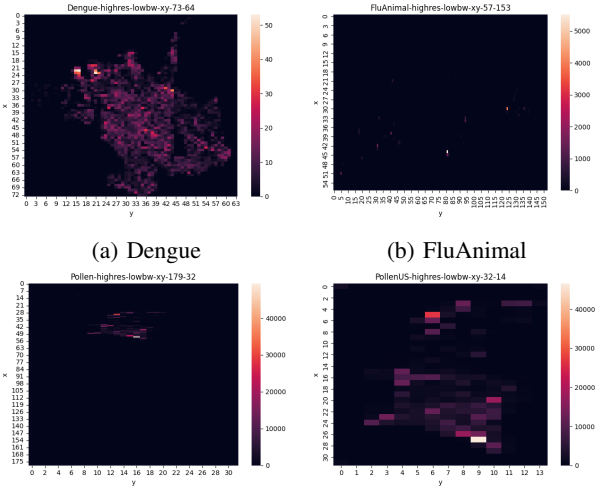
A. Experimental Setting

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 [4]. 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 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.



(a) Dengue (b) FluAnimal

(c) Pollen (d) PollenUS

Fig. 4: Instance Samples

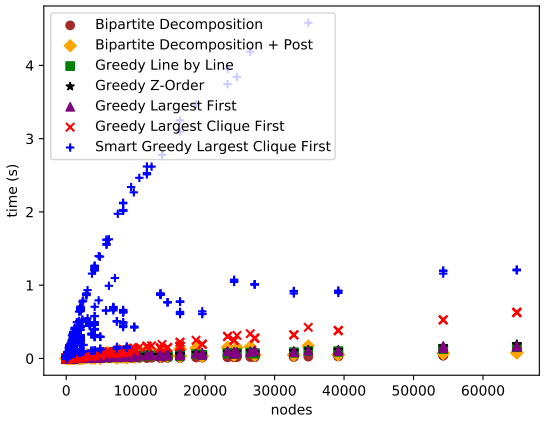
B. 2D Results

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 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 5a and 5b, respectively. Performance profiles broken down by 2D dataset are shown in Figure 6.

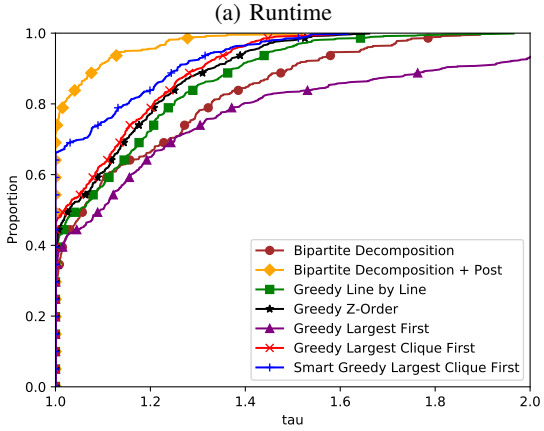
In general, BDP performed substantially better than all other algorithms. On average, BDP obtained a solution within 1.03 times the lower bound of maximum K_4. BDP was 182% faster than SGK and required 1.69% less colors. BDP and SGK yielded the highest percentage of solutions that can be proven optimal 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.

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 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



(a) Runtime



(b) Performance Profile: *maxcolor*

Fig. 5: 2D Results (All Instances)

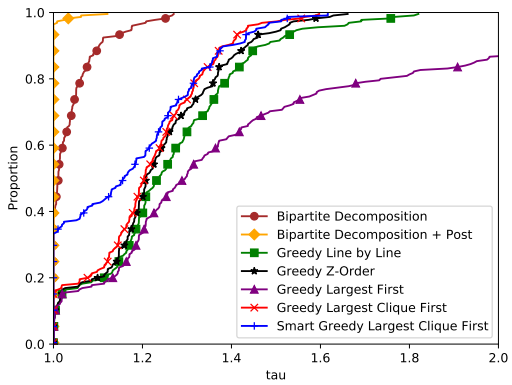
processing was 136% slower on average; however, this number may be skewed. In many cases, BD obtained a solution faster than it could be measured. Thus, the wallclock used to measure time returned a value of 0.

C. 3D Results

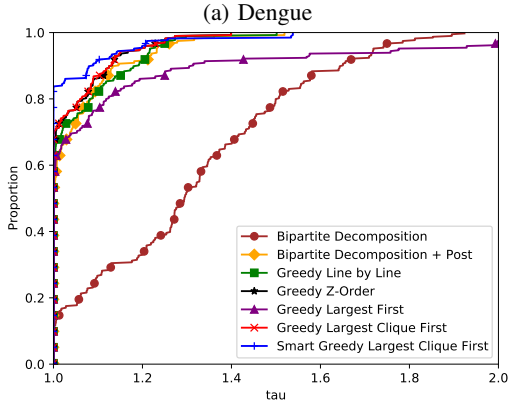
The runtime comparison and performance profile for 3D instances can be found in Figures 7a and 7b, respectively. Performance profiles broken down by 3D dataset are shown in Figure 8.

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.

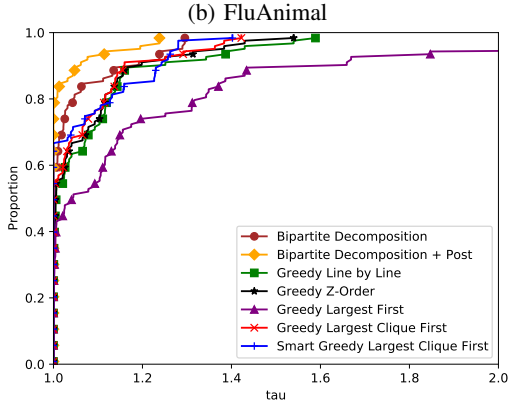
BDP had a mediocre performance on 3D instances, whereas it was the clear favorite on the 2D instances. In 3D, BDP obtained an average *maxcolor* with a higher than average runtime. Furthermore, the different 3D instances seemed to have a greater impact on algorithm performance than in the 2D cases.



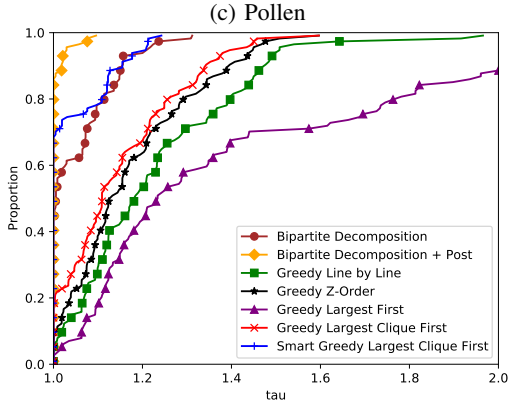
(a) Dengue



(b) FluAnimal

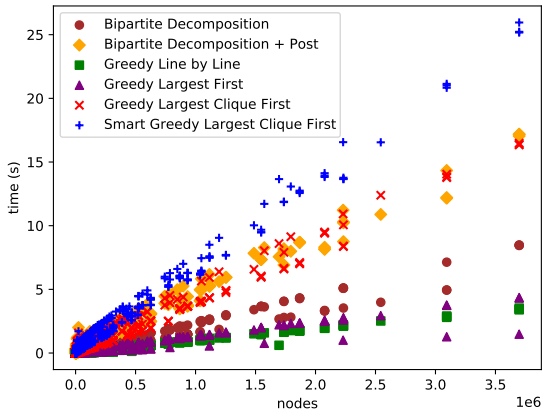


(c) Pollen

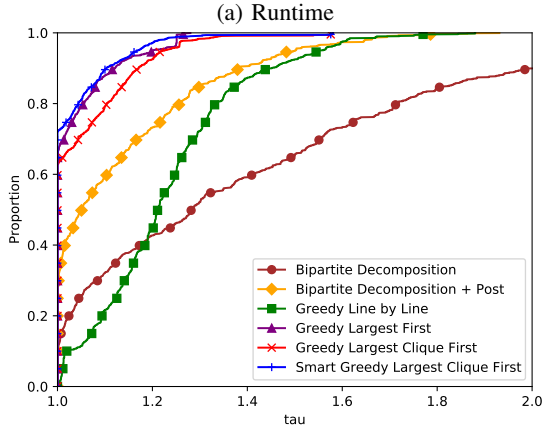


(d) PollenUS

Fig. 6: Performance Profile for 2DS-IVC: *maxcolor* broken down per dataset



(a) Runtime



(b) Performance Profile: $maxcolor$

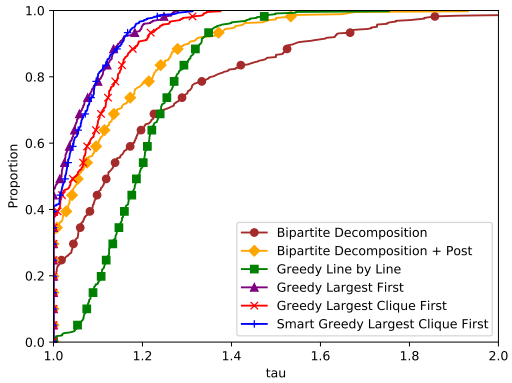
Fig. 7: 3D Results (All Instances)

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.

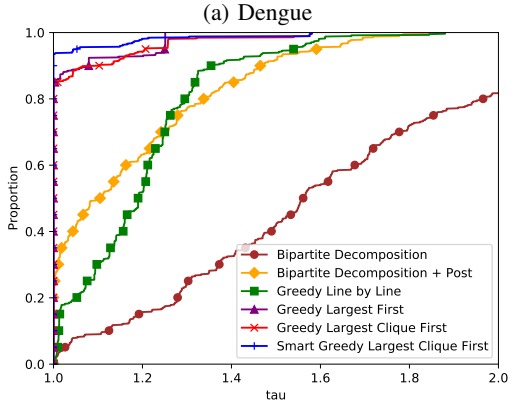
We would also expect to see the 2D results upheld in instances where consecutive vertices in the sequence of largest weights appear on the same plane. The results seem to reflect this argument: 18.1% of 3D instances BDP performs strictly better than SGK in terms of $maxcolor$. We conclude that specific 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.

D. Optimal coloring based analysis

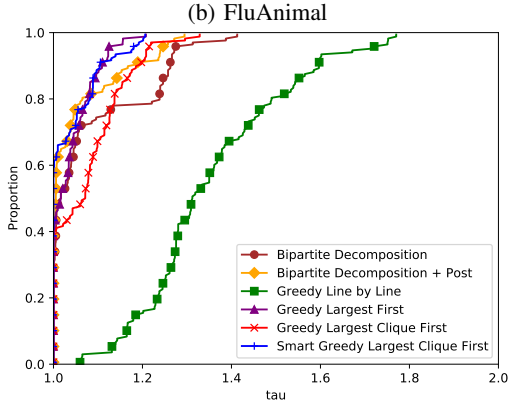
In order to further analyze the performance of the heuristics, we designed a Mixed Integer Linear Program (MILP) and solved optimally as many instanes as we could. We solved the MILP using Gurobi and let the solver run for one day per instance on a node of a computing cluster. Most of the instances were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



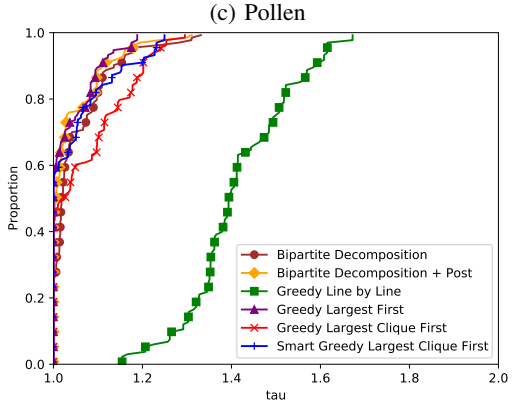
(a) Dengue



(b) FluAnimal

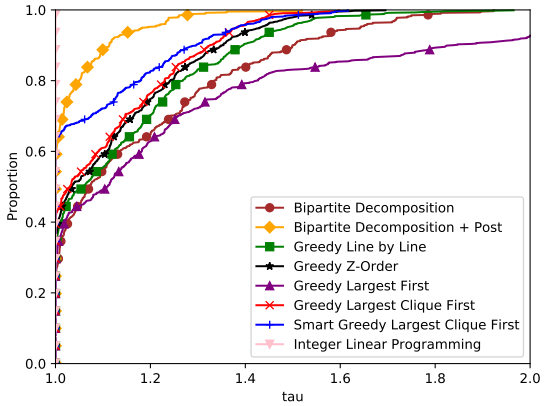


(c) Pollen

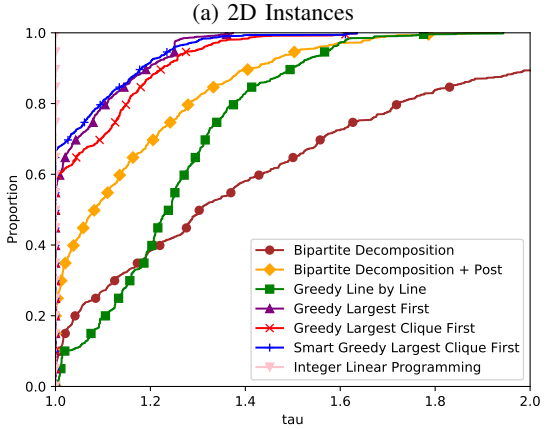


(d) PollenUS

Fig. 8: Performance Profile on 3DS-IVC: $maxcolor$ broken down by dataset



(a) 2D Instances



(b) 3D Instances

Fig. 9: Performance Profiles with ILP

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 9a and 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.

VII. 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 [4]. In this application, some events (points) are located in a 3D space and the space is discretized in voxels. The computational load is 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 [4].

The application has many modes of parallelisation 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 [4]. 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 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

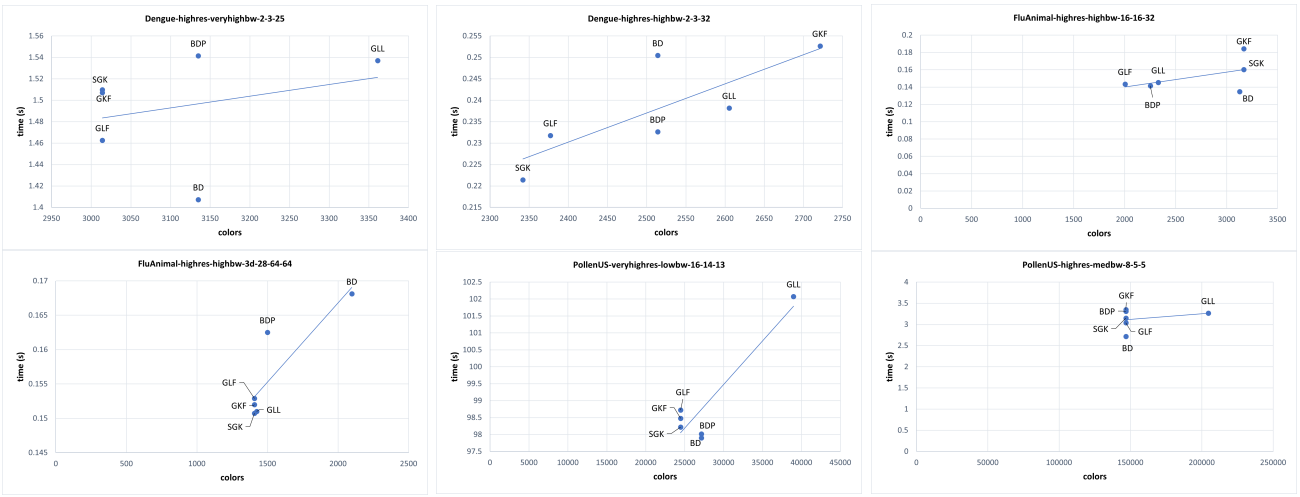


Fig. 10: Scatter plot of number of colors and execution time of the STKDE application. Each scatter plot presents different coloring algorithm. A linear regression line shows positive corelation between number of colors and runtime in all 6 cases.

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.

VIII. 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?

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No CCF-1652442.

References

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

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

[3] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *SIGGRAPH Computer Graphics*, vol. 21, no. 4, p. 25–34, Aug. 1987.

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

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

[6] M. Bouchard, M. Čangalović, 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.

[7] 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.

[8] 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.

[9] D. de Werra and A. Hertz, "Consecutive colorings of graphs," *Zeitschrift für Operations Research*, vol. 32, no. 1, pp. 1–8, Jan 1988.

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

[11] 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.

[12] 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.

[13] 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.

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

[15] B. M. E. Moret, "Planar NAE3SAT is in P," *SIGACT News*, vol. 19, no. 2, p. 51–54, Jun. 1988.