Component Coloring of Proper Interval Graphs and Split Graphs

Ajit Diwan, Soumitra Pal, Abhiram Ranade
Department of Computer Science and Engineering,
Indian Institute of Technology Bombay,
Powai, Mumbai 400076, India.
{aad,mitra,ranade}@cse.iitb.ac.in

Abstract

We introduce a generalization of the well known graph (vertex) coloring problem, which we call the problem of component coloring of graphs. Given a graph, the problem is to color the vertices using minimum number of colors so that the size of each connected component of the subgraph induced by the vertices of the same color does not exceed $C$. We give a linear time algorithm for the problem on proper interval graphs. We extend this algorithm to solve a weighted version of the problem in which vertices have integer weights and can be split into differently colored parts, so that the total weight of a monochromatic component does not exceed $C$. We also prove that the problem is NP-hard for split graphs.

1 Introduction

The vertex coloring problem is to color the vertices of a graph using minimum number of colors so that no two adjacent vertices are assigned the same color. In this paper, we introduce and study a generalization of the vertex coloring problem. In this generalized problem, called the problem of component coloring of graphs, we allow two adjacent vertices to be assigned the same color. It is customary to consider two variations: weighted and unweighted. The weighted version of the problem is defined as follows. Given a graph $G(V,E)$ and for each $v \in V$ a weight $0 < w_v \leq 1$, color the vertices using minimum number of colors such that the total weight of any monochromatic component, i.e., the connected component of the subgraph induced by the vertices of the same color, does not exceed 1.

If the weights of all the vertices are same and equal to $1/C$, where $C$ is a positive integer, we get an unweighted version of the problem where the objective is to use minimum number of colors so that size of each monochromatic component in the coloring is at most $C$. The vertex coloring is a special case of the unweighted component coloring problem where $C = 1$ and each monochromatic component consists of a single vertex. Since the vertex coloring problem is NP-hard on general graphs \cite{1}, the unweighted (and hence weighted) component coloring problem is also NP-hard on general graphs.

A graph $G$ is an interval graph if there exists a family $I$ of intervals in a linearly ordered set (like the real line), and there exists a one-to-one correspondence between the vertices of $G$ and the intervals in $I$ such that two vertices are adjacent if and only if the corresponding intervals intersect. If no interval of $I$ properly contains another, set theoretically, then $G$ is called a proper interval graph.

When $C = 1$ the problem, i.e., the vertex coloring problem, has a polynomial time algorithm on interval graphs \cite{2}. However, the complexity of the problem for general $C$ on interval graphs, is not known. In this paper we give a polynomial time algorithm for general $C$ on proper interval graphs.

The vertex coloring problem also has polynomial time algorithm for split graphs, i.e., when the vertex set can be partitioned into an independent set and a clique \cite{3}. We prove that the problem is NP-hard for general $C$ on split graphs.

We also consider a splittable weighted version of the problem in which each vertex of the input graph has an integer weight which can be distributed among multiple copies of the vertex. These copies can be colored separately. However, the total weight of a monochromatic component in the resultant graph should not exceed $C$. We extend the algorithm for the unweighted problem to solve this splittable weighted problem on proper interval graphs.
1.1 Applications

Our formulation of the component coloring problem is motivated by a problem on scheduling transmission requests on light-trails, a hardware solution for bandwidth provisioning in optical WDM (Wavelength Division Multiplexing) networks [4]. In a path network of processors using light-trails, each processor has an optical shutter for each wavelength which can be configured to be switched on/off for allowing/blocking the light signal pass through it. For each wavelength, by suitably configuring the optical shutter at each processor, the logical path network can be partitioned into subpath networks in which multiple transmissions can happen in parallel, provided the total bandwidth requirement of the transmissions assigned to a subpath does not exceed the capacity of a wavelength. Such subpaths, in which only the end processors have their optical shutters blocked, are called light-trails. A light-trail can serve only the transmissions having both source and destination within the light-trail. If a transmission is assigned to a light-trail, it uses the complete physical span of the light-trail. Given a set of transmission requests, the scheduling problem is to configure the optical shutters at the processors so that minimum number of wavelengths are required by the light-trails to serve all transmission requests.

The light-trail scheduling problem on path networks can be posed as a component coloring problem on interval graphs as follows. For each transmission request, create a vertex with weight equal to the bandwidth requirement, expressed as a fraction of the wavelength capacity. Two vertices are adjacent if the corresponding transmissions overlap, i.e., they use at least one common link. Given a solution to the component coloring problem, a solution to the light-trail scheduling problem can be constructed as follows. For each of the used colors, use a separate wavelength. For each wavelength, construct a separate light-trail for each monochromatic component of the corresponding color. Note that the light-trails on a wavelength do not intersect with each other. All transmission requests corresponding to the vertices of a monochromatic component are served by the corresponding light-trail. The physical span of the light-trail is the union of the physical spans of all requests in it. For each wavelength, the optical shutters in the processors at the endpoints of all light-trails on the wavelength are configured to be off.

As mentioned in [5], the light-trail scheduling problem is similar to the problem of scheduling in reconfigurable bus architectures [6,7] and hence component coloring applies there too.

1.2 Related Work

The NP-hard light-trail scheduling problem with arbitrary bandwidth requirements on ring networks and general networks has generally been solved using heuristics [8,9,10,11,12,13,14,15]. However, these solutions do not provide any bound on the performance. For path networks and ring networks, [5] gives an approximation algorithm that uses $O(w + \log p)$ wavelengths where $p$ is the number of processors in the network and $w$ is the congestion, i.e., the maximum total traffic required to pass through any link, and hence a lower bound on the number of wavelengths used. For the scheduling problem on reconfigurable bus architectures, the only theoretical work that we know, deals with random transmission patterns using standard techniques such as Chernoff bounds [16]. However, in our knowledge, there is no work in literature to solve the problem exactly for the special cases such as when no transmission uses a set of links that is a proper subset of links used by another transmission and/or when all transmissions have equal bandwidth requirements.

In the graph coloring literature, there are works to solve a problem that is a kind of dual to the unweighted component coloring problem [17,18]. Here, the objective is to minimize the size of the largest monochromatic component in a coloring using a fixed number of colors. The paper [17] shows that for a $n$-vertex graph of maximum degree 4, there exists an algorithm that uses 2 colors and produces a coloring in which the size of the largest monochromatic component is $O(2^{(2\log_2 n)^{1/2}})$. For a family of minor-closed graphs, [18] shows that if $\lambda$ colors are used, the size of the largest monochromatic component is in between $\Omega(2^{\lambda(2\lambda-1)})$ and $O(2^{\lambda(\lambda+1)})$ for every fixed $\lambda$.

1.3 Our Results

Theorem 1. There exists an algorithm that solves the unweighted component coloring problem on a proper interval graph $G = (V, E)$ in $O(|V| + |E|)$ time.

Theorem 2. There exists an algorithm that solves the splittable weighted component coloring problem on a proper interval graph $G = (V, E)$ in $O(|V| + |E|)$ time.

Theorem 3. The unweighted component coloring problem is NP-hard for split graphs.
The rest of the paper is organized as follows. In Section 2 we present some pertinent definitions and known results. In Section 3 we show that for the class of chordal graphs, the component coloring problem is equivalent to a vertex partitioning problem. Note that the interval graphs and the split graphs are chordal. We give the details of the solution to the partitioning problem on proper interval graphs in Section 4, spanning across several subsections. In Subsection 4.1 we show that for proper interval graphs, the constraints of the partitioning problem can be restricted further by letting all vertices in a part be consecutive in some linear order of the vertices. Utilizing the lower bound and the upper bound for the restricted partitioning problem, given in Subsections 4.2 and 4.3 respectively, we give an LP based solution in Subsection 4.4. We give a combinatorial algorithm for the same problem in Subsection 4.5. In Subsection 4.6 we extend this algorithm to solve the splittable weighted problem. We prove the NP-hardness of the problem on split graphs in Section 5.

2 Preliminaries

Throughout this paper, let $G = (V, E) = (V(G), E(G))$ be a simple, undirected graph and let $n = |V|$ and $m = |E|$. We also assume that $G$ is connected. If $G$ is not connected, the results in this paper can be applied separately to each of its connected components. For a set $S \subseteq V$, the sub-graph of $G$ induced by $S$ is $G[S] = (S, E(S))$ where $E(S) = \{(u, v) \in E \mid u, v \in S\}$. For a set $S \subseteq V$, $V - S$ denotes $G[V \setminus S]$. A clique of $G$ is a set of pair wise adjacent vertices of $G$. The size of a clique is the number of vertices in it. A maximal clique is a clique of $G$ that is not properly contained in any clique of $G$. A maximum clique is a clique of maximum size. The clique number of $G$, denoted by $\omega(G)$ or simply $\omega$, is the size of a maximum clique of $G$. An independent set of $G$ is a set of pairwise non-adjacent vertices in it. Coloring of a graph is an assignment of colors to its vertices. A $\lambda$-assignment of a graph $G = (V, E)$ is a map from $V$ to some set of $\lambda$ colors such as $\{1, \ldots, \lambda\}$; this assignment may not be 'proper' in the standard notion of graph (vertex) coloring that two adjacent vertices must be assigned different colors. A color class $i$ is the set of vertices assigned color $i$ under the $\lambda$-assignment. A monochromatic component of $G$ under a $\lambda$-assignment is a component of the sub-graph induced by a single color class, or in other words, a maximal connected monochromatic sub-graph. Following the terminology of [17], we call a monochromatic component a chromon. The size of a chromon is the number of vertices in it. A $k$-chromon is a chromon of size $k$.

A graph is $[\lambda, C]$-colorable if it has a $\lambda$-assignment in which every chromon has at most $C$ vertices and such an assignment is called a $[\lambda, C]$-coloring. A $C$-component coloring of graph $G$ is a $[\lambda, C]$-coloring with minimum $\lambda$. If there is no ambiguity in the context, we will omit $C$ and use the term component coloring to denote $C$-component coloring. Sometimes we will simply refer to the problem as the coloring problem. Note that the vertex coloring is to find a 1-component coloring in which chromons are just the individual vertices. The component coloring problem can also be seen as a problem of partitioning the vertex set such that each part is a chromon. However, we will study a slightly different partition problem which we will show to be equivalent to the coloring problem for some classes of graphs. A graph $G = (V, E)$ is said to have a $[\lambda, C]$-partition if and only if there is a partition $\Pi = \{P_1, P_2, \ldots, P_t\}$ of $V$, $P_i \subseteq V$, $P_i \cap P_j = \emptyset$ for all $i \neq j$ such that the following constraints are satisfied: 1. connectedness – the sub-graph induced by each part $P_i$, i.e., $G[P_i]$ is connected, 2. size – each part $P_i$ has at most $C$ vertices, and 3. clique intersection – any clique in $G$ intersects at most $\lambda$ parts ($\lambda$ will subsequently be called the clique intersection of the partition). A $C$-component partition of a graph is a $[\lambda, C]$-partition with minimum clique intersection $\lambda$. We will refer to the problem of finding a $C$-component partition as the partition problem.

We study the coloring problem on interval graphs and split graphs. Each of these classes of graphs is a subclass of the class of chordal graphs. A graph is chordal if each of its cycles of four or more vertices has a chord, which is an edge joining two vertices that are not adjacent in the cycle. There are many characterizations of chordal graphs (see [3] for more details). We will use the characterization of a chordal graph as an intersection graph. Note that a graph is an intersection graph if the vertices correspond to the subsets of a set, and two vertices are adjacent if the corresponding subsets have at least one common element.

**Proposition 4 ([3]).** Let $G = (V, E)$ be an undirected graph. The following statements are equivalent:

1. $G$ is a chordal graph.
2. $G$ is the intersection graph of a family of subtrees of a tree.
3. There exists a tree $T = (Q, E)$ whose vertex set $Q$ is the set of maximal cliques of $G$ such that for each $v \in V$ the induced subgraph $T[Q_v]$ is connected (and hence a subtree), where $Q_v$ consists of those maximal cliques that contain $v$. 

3
Statement 3 of Proposition 4 gives a representation of a chordal graph $G$ using intersection of subtrees of a tree $T$ in which the vertices of $T$ are the maximal cliques of $G$. Since a chordal graph is the intersection graph of a family of subtrees of a tree, the following result on the intersections of subtrees of a tree will be useful in the context of component coloring of chordal graphs:

**Proposition 5** ([2]). A family of subtrees $\{T_i \mid i \in I\}$ of a tree $T$ satisfies the Helly property, i.e., for $J \subseteq I$ and $T_i \cap T_j \neq \emptyset$ for all $i, j \in J$ implies $\cap_{j \in J} T_j \neq \emptyset$. □

A graph $G = (V, E)$ is a split graph if there is a partition $V = S + Q$ of its vertex set into an independent set $S$ and a clique $Q$. There is no restriction on edges between vertices of $S$ and $Q$. Since every alternate vertex of a cycle in $G$ must be from $Q$, there can not be a cycle of four or more vertices without a chord. Thus $G$ is also chordal.

A graph $G = (V, E)$ is an interval graph if there exists a family $I = \{I_v \mid v \in V\}$ of intervals on a real line such that for distinct vertices $u, v$ in $G$, $(u, v) \in E$ if and only if $I_u \cap I_v \neq \emptyset$. Such a family $I$ of intervals is commonly referred to as the interval representation of $G$. Given an interval representation of $G$, consider only the set $A$ of endpoints of the intervals. Note that these endpoints follow a linear order (as on the real line). Now consider a graph $T$ having a vertex corresponding to an element in $A$, and there is an edge between two vertices in $T$ if the corresponding endpoints are adjacent on the linear order. Then $T$ is a path and $G$ is the intersection graph of subpaths of $T$ where each subpath corresponds to an interval. Since a path is also a tree, $G$ is also chordal. It will be convenient to let $Left(I_v)$ and $Right(I_v)$ stand for the left and right endpoint of the interval $I_v$, respectively. The family $I$ is the interval representation of a proper interval graph (PIG) if and only if no interval is properly contained in another. For a given interval graph $G$ we can get an interval representation of it using the following result:

**Proposition 6** ([19]). A graph $G$ is an interval graph if and only if its maximal cliques can be linearly ordered such that for every vertex $v$ of $G$, the cliques containing $v$ occur consecutively in the linear order. □

Let $\bar{Q}_1, \bar{Q}_2, \ldots, \bar{Q}_t$ be an ordering of the maximal cliques of an interval graph $G$ as in Proposition 6. For every vertex $v$ of $G$, let $I_v$ stand for the set $\{Q_i \mid v \in \bar{Q}_i\}$. Then by Proposition 6 $I_v$ is an interval. Hence $\{I_v \mid v \in V\}$ is an interval representation for $G$. We shall refer to this as max-clique representation.

**Proposition 7** ([2]). A graph $G = (V, E)$ is an interval graph if and only if there exists a linear order $<$ on $V$ such that for every choice of vertices $u, v, w$ with $u < v < w$, $(u, w) \in E$ implies $(u, v) \in E$. □

Consider an interval graph $G = (V, E)$ and an interval representation $I = \{I_v \mid v \in V\}$ of $G$. As in the proof of Proposition 7 in [2], consider the linear order $<$ on $V$ defined as follows. For $u, v \in V$, $u < v$ if and only if $Left(I_u) < Left(I_v)$ or $(Left(I_u) = Left(I_v))$ and $(Right(I_u) \leq Right(I_v))$. We call this ordering $v_1 < v_2 < \cdots < v_n$ the canonical ordering. In the rest of the paper, we use numbers 1 to $n$ to represent the vertices where $i$ represents the vertex that appears $i$th in the canonical ordering. Hence, $v$ will be interchangeably used to represent a vertex $v \in V$ as well as its position in the canonical ordering. If $u < v$ then $u$ is said to be on the left of $v$ and $v$ is said to be on the right of $u$.

For PIGs the canonical ordering not only satisfies the conditions in Proposition 7 but, in fact, satisfies a stronger property:

**Proposition 8** (“The Umbrella Property” [20]). A graph $G = (V, E)$ is a PIG, if and only if, there exists a linear order $<$ on $V$ such that for every choice of vertices $u, v, w$, with $u < v < w$, $(u, w) \in E$ implies both $(u, v) \in E$ and $(v, w) \in E$. □

**Corollary 9.** Any maximal clique of a PIG has vertices that are consecutive in the canonical ordering and hence can be represented by a single edge between the two end vertices; other edges immediately follow from the umbrella property. □

**Corollary 10.** Let $S$ be a connected component of a PIG and $v_1, v_2, \ldots, v_t$ be the vertices of $S$ arranged in the canonical ordering. Then there must be an edge between $v_i$ and $v_{i+1}$ for all $i = 1, 2, \ldots, t-1$.

Proof. Consider the two vertices $v_i$ and $v_{i+1}$. Suppose there is no edge between them. Since $S$ is connected there must be a path $v_i \sim v_j \sim v_k \sim v_{i+1}$ where $j < i$ and $i + 1 < k$. Since there is an edge between $v_j$ and $v_k$ and $v_{i+1} < v_k$, by the umbrella property, there is an edge between $v_i$ and $v_k$. Since $v_i < v_{i+1} < v_k$, again by the umbrella property there is an edge between $v_i$ and $v_{i+1}$. This is a contradiction. □
3 Equivalence of Coloring and Partition on Chordal Graphs

Lemma 11. If a graph \( G \) has a \( [\lambda, C] \)-coloring then it has a \( [\lambda, C] \)-partition.

Proof. Suppose \( G \) has a \( [\lambda, C] \)-coloring \( C \). Consider the partition \( \Pi \) induced by \( C \) where each part is exactly a chromon. The connectedness constraint is immediately satisfied. Since a chromon has size at most \( C \), the size constraint is also satisfied. Since any pair of vertices in a chromon is directly connected by an edge, the chromons in \( C \) intersected by a clique are all of different colors. Hence, a clique intersects at most \( \lambda \) parts in \( \Pi \). Thus the clique intersection constraint is also satisfied. Hence, \( \Pi \) is a \( [\lambda, C] \)-partition. \( \square \)

Next we will show that for chordal graphs the converse is also true.

Lemma 12. If a chordal graph \( G \) has a \( [\lambda, C] \)-partition then it has a \( [\lambda, C] \)-coloring.

Proof. Suppose \( G \) has a \( [\lambda, C] \)-partition \( \Pi = \{P_1, P_2, \ldots, P_t\} \). Now consider the representation \( T = (Q, E) \) whose vertex set \( Q \) is the set of maximal cliques of \( G \) as given in statement 3 of Proposition 4. Each vertex in \( G \) is represented by a subtree in \( T \). Since \( P_i \) is connected, the union of all subtrees corresponding to the vertices in \( P_i \) is also a subtree. Let us call this subtree \( T_i \). By Proposition 1 the collection of \( T_i \)'s represent another chordal graph \( G' = (V', E') \), where each vertex in \( v' \in V' \) correspond to some part \( P_i \) in \( \Pi \). Note that \( G \) and \( G' \) have the same clique tree representation \( T \), only the subtrees corresponding to the vertices of the two graphs differ.

If we can get a vertex coloring \( C \) for \( G' \) using \( \lambda \) colors then we are done. We get a \( [\lambda, C] \)-coloring for \( G \) by assigning the vertices belonging to a part \( P_i \) the color of the corresponding vertex \( v'_i \) in the coloring \( C \) of \( G' \).

Since a chordal graph has clique number equal to the chromatic number, and there is a polynomial time algorithm 3 to get a vertex coloring of a chordal graph, it is enough to show that clique number of \( G' \) is \( \lambda \).

Consider a clique \( Q' = \{v'_{i_1}, v'_{i_2}, \ldots, v'_{i_k}\} \) of size \( k \) in \( G' \). Since the subtrees \( T_i \)'s corresponding to the vertices in \( Q' \) are mutually intersecting, by Helly property (Proposition 4), there must be a vertex \( T_Q \) in \( T \) which is common to all subtrees \( T_{i_1}, T_{i_2}, \ldots, T_{i_k} \). This represents a maximal clique \( Q \) in \( G \). Since \( T_Q \) is common to all \( T_{i_1}, T_{i_2}, \ldots, T_{i_k} \), \( Q \) intersects \( k \) parts \( P_{i_1}, P_{i_2}, \ldots, P_{i_k} \). But \( k \) can be at most \( \lambda \). Hence the size of the maximum clique in \( G' \) is \( \lambda \). \( \square \)

Henceforth, to prove that a chordal graph \( G \) has a \( [\lambda, C] \)-coloring, we just show that \( G \) has a \( [\lambda, C] \)-partition. In fact, we solve the partition problem only because the solution can be converted to a solution to the coloring problem using the procedure described in Lemma 12.

4 Partition on Proper Interval Graphs

For PIGs, we introduce a more restricted way of partitioning the vertex set. We first define a new term. A block in a PIG is a set of vertices which are consecutive in the canonical ordering. A PIG is said to have a block partition if it has a partition in which each part also satisfy consecutiveness constraint, i.e., each part is also a block.

We illustrate the differences between coloring, partition and block partition using an example in Fig. 1. This example shows that there are instances where both \( [\lambda, C] \)-partition and \( [\lambda, C] \)-block partition exist. Is this true for all instances? The answer turns out to be true which we prove next.

4.1 Equivalence of Partition and Block Partition

Lemma 13. If a PIG \( G \) has a \( [\lambda, C] \)-partition then it has a \( [\lambda, C] \)-block partition.

Proof. Suppose \( G \) has a \( [\lambda, C] \)-partition \( \Pi \). If the parts in \( \Pi \) also satisfy the consecutiveness constraint, we are done. Otherwise, we convert \( \Pi \) to a new partition \( \Pi' \) that also satisfies the consecutiveness constraint. The conversion is done by exchanging vertices among the parts in \( \Pi \), step-by-step, as follows.

We call a vertex \( u \) to be separated if \( u, u+1 \) and \( v > u+1 \) are in some clique \( Q \) and \( u \) and \( v \) belong to one part but \( u+1 \) belongs to a different part. Suppose, at the beginning of a step, \( i \) is the leftmost separated vertex such that \( i \) belongs to a part \( P_1 \), but \( i+1 \) belongs to a different part \( P_2 \). Such an \( i \) exists because we assumed that the consecutiveness constraint is not satisfied. Our aim is to interchange vertices between \( P_1 \) and \( P_2 \) such that \( i \) becomes non-separated without making any vertex on the left of \( i \) separated. Then by repeating this process all vertices can be made non-separated and hence consecutiveness constraint will be satisfied. We replace \( P_1 \) and \( P_2 \) in \( \Pi \) by \( P'_1 \) and \( P'_2 \) to get \( \Pi' \) as follows.

\(^1\)Some authors use the term block to represent what we call a clique.
There is an edge between at most \(P_1\) and \(P_2\), i.e., if \(|P| \leq C\), \(P'_2\) contains all vertices in \(P\), and \(P'_1\) contains the vertices of \(P_1 \cup P_2\) that are not included in \(P'_2\); otherwise \(P'_2\) contains the rightmost \(C\) vertices of \(P\), and \(P'_1\) contains the vertices of \(P_1 \cup P_2\) that are not included in \(P'_2\). See Fig. 2 for examples of both cases.

Let \(P\) be the set of vertices in \(P_1 \cup P_2\) that are on the right of \(i\). We put at most \(C\) rightmost vertices from \(P\) to \(P'_2\) and remaining from both \(P_1\) and \(P_2\) to \(P'_1\), i.e., if \(|P| \leq C\), \(P'_2\) contains all vertices in \(P\), and \(P'_1\) contains the vertices of \(P_1\) that are not included in \(P'_2\); otherwise \(P'_2\) contains the rightmost \(C\) vertices of \(P\), and \(P'_1\) contains the vertices of \(P_1 \cup P_2\) that are not included in \(P'_2\). See Fig. 2 for examples of both cases.

Since \(i\) is the leftmost separated vertex in \(\Pi\), \(i + 1\) is the leftmost vertex of \(P_2\). So \(P'_1\) either has \(i\) as the rightmost vertex or has both \(i\) and \(i + 1\). Thus \(i\) is no more a separated vertex in \(\Pi\). The sizes of the parts in \(\Pi\) do not exceed \(C\). Let \(j\) be the next vertex on the right of \(i\) in \(P_1\). Such a vertex exists because \(i\) is separated. Since \(P_1\) is connected, by Corollary 10 there is an edge between \(i\) and \(j\). Since \(i < i + 1 < j\), by the umbrella property, there is an edge between \(i\) and \(i + 1\). Since \(P_2\) is also connected, \(P_1 \cup P_2\) is connected. By Corollary 10 there is an edge between every two near by vertices in \(P_1 \cup P_2\). Since the vertices of \(P_1 \cup P_2\) are distributed in \(P'_1\) and \(P'_2\) such that all vertices of \(P'_i\) are on the left of all vertices in \(P'_2\). Thus \(P'_1\) and \(P'_2\) are connected. So connectedness constraint is satisfied. It only remains to show that the clique intersection constraint is not violated in \(\Pi\). We do so by contradiction as follows.

Suppose the clique intersection constraint is violated in \(\Pi'\). Then there is a clique \(Q\) which intersects at least \(\lambda + 1\) parts in \(\Pi'\). Without loss of generality, we assume that \(Q\) is a maximal clique because otherwise we can consider the maximal clique that includes \(Q\). Since clique intersection constraint is satisfied in \(\Pi\), \(Q\) intersects at most \(\lambda\) parts in \(\Pi\). The exchange does not affect the cliques which intersect neither \(P_1\) nor \(P_2\). Also, clique intersection can increase by at most 1. Hence \(Q\) must be such that it intersects \(\lambda + 1\) blocks in \(\Pi'\) including both \(P'_1\) and \(P'_2\), but intersects \(\lambda\) parts in \(\Pi\) including exactly one of \(P_1\) and \(P_2\). By Corollary 9 \(Q\) has consecutive vertices and it intersects both \(P'_1\) and \(P'_2\). Hence it must include the rightmost vertex of \(P'_1\) and the leftmost...
vertex of $P'_2$. But $Q$ intersects only one of $P_1$ and $P_2$. Hence $i$ cannot be the rightmost vertex in $P'_1$. As per our construction, this implies $|P'_1| = C$. We show a contradiction in each of the following two possible cases.

Case 1: $Q$ intersects $P_1$ only (see Fig. 3(a)). Since $Q$ does not include $i + 1$, it must be on the right of $i + 1$. Let $u$ be the rightmost vertex of $P_1$ that is on the left of $Q$. There must be at least one vertex of $P_2$ that is on the right of $Q$. Because otherwise $Q$ contains all $C$ vertices of $P'_2$ which also belong to $P_1$. Since $i$ also belongs to $P_1$, this makes the size of $P_1$ exceed $C$ which is not possible. Let $v$ be the leftmost vertex in $P_2$ on the right of $Q$. Since $P_2$ is connected, there must be a direct edge between $u$ and $v$. Hence $u$ and $v$ belong to a clique $Q' \supseteq Q$. Thus $Q' \cap Q$ intersects $\lambda + 1$ parts in $\Pi$. This is a contradiction.

Case 2: $Q$ intersects $P_2$ only. Since $Q$ does not include $i$, it must be on the right of $i$. Let $u$ be the rightmost vertex of $P_1$ that is on the left of $Q$. We claim that there must be at least one vertex of $P_1$ that is on the right of $Q$. This is obvious if $Q$ is on the left of $j$ (see Fig. 3(b)). Otherwise $Q$ is on the right of $j$ (see Fig. 3(c)). If there is no vertex from $P_1$ on the right of $Q$, then $Q$ contains all $C$ vertices of $P'_2$ which also belong to $P_2$. Since $i + 1$ also belong to $P_2$ and does not belong to $P'_2$, this makes the size of $P_2$ exceed $C$ which is not possible. Let $v$ be the leftmost vertex in $P_2$ on the right of $Q$. Since $P_1$ is connected, there must be a direct edge between $u$ and $v$. Hence $u$ and $v$ belong to a clique $Q' \supseteq Q$. Thus $Q' \cap Q$ intersects $\lambda + 1$ parts in $\Pi$. This is a contradiction.

Henceforth, to prove that a PIG $G$ has a $[\lambda, C]$-partition, we just show that $G$ has a $[\lambda, C]$-partition. In fact, in the rest of this section we will abuse the notion $[\lambda, C]$-partition to actually mean $[\lambda, C]$-block partition.

### 4.2 Lower Bound

**Lemma 14.** Let $\omega$ be the clique number of a PIG $G$. If $G$ has a $[\lambda, C]$-partition then $\lambda \geq \lceil \omega / C \rceil$.

**Proof.** Let $Q$ be a maximum clique of $G$ having a $[\lambda, C]$-coloring. Consider the vertices of $Q$ only. To cover all $\omega$ vertices of $Q$ by parts of size at most $C$, we need at least $\lceil \omega / C \rceil$ parts. For any $S \subseteq Q$, $G[S]$ is connected. Hence, number of parts intersected by a clique is $\lambda \geq \lceil \omega / C \rceil$.

There is a simple counter example where $\lambda = \lceil \omega / C \rceil$ is not enough to have a $[\lambda, C]$-partition. Consider $G(V, E)$ where $V = \{a, b, c\}$ and $E = \{(a, b), (b, c)\}$. It has a proper interval representation $I_a = \{1, 3\}; I_b = \{2, 5\}; I_c = \{4, 6\}$. Here, $\omega = 2$. For $C = 2$, $\omega / C = 1$ is not enough as the three vertices belong to the same connected component.

### 4.3 Upper Bound

Consider the following greedy algorithm for the partitioning problem for a connected PIG $G$.

**Lemma 15.** Let $\omega$ be the clique number of a connected PIG $G$. Then **GreedyPartition** produces a valid $[(\omega + C - 1) / C, C]$-partition in $O(m + n)$ time.
Lemma 16. If there is an \( \lambda \)-component partition problem for any PIG.\footnote{For \( \lambda \) integer and \( 2 \leq \lambda \leq \omega \), the two bounds are \( \lambda \leq \omega \) and \( \lambda \geq \omega - 1 \).} The greedy procedure takes time \( O(n) \). Overall it takes \( O(m + n) \) time. \qed

Thus the \( \lfloor \lambda, C \rfloor \)-partition found by an optimal algorithm for the \( C \)-component partition problem has \( \lambda \) no more than \( \lfloor (\omega + C - 1)/C \rfloor \). There is a simple example where the minimum value of \( \lambda \) is strictly less than this upper bound. Consider \( G(V, E) \) where \( V = \{a, b, c, d\} \), and \( E = \{(a, b), (c, d)\} \). It has a proper interval representation \( I_a = [1, 3]; I_b = [2, 4]; I_c = [5, 7]; I_d = [6, 8] \). Here, \( \omega = 2. \) For \( C = 2, \lfloor (\omega + C - 1)/C \rfloor = 2 \) but the partition \( \{\{a, b\}, \{c, d\}\} \) is a \( [1, 2] \)-partition and hence minimum value of \( \lambda \) is \( 1 \).

Let us now consider the following special case of the partition problem: given a PIG with \( \omega = kC + r \), \( k \) integer and \( 2 \leq r \leq C \), check if there is a \( [k + 1, C] \)-partition and if so, generate the partition. We call this the partition subproblem.

Lemma 16. If there is an \( O(m + n) \) time algorithm for the partition subproblem, then there is an \( O(m + n) \) time algorithm for the partition problem for any PIG.\footnote{For \( \omega \equiv 1 \mod C \), the lower bound given in Lemma 13 and the upper bound given in Lemma 15 match and \textsc{Greedy Partition} gives an optimal solution to the \( C \)-component partition problem in \( O(m + n) \) time.}

4.4 An LP Based Algorithm

The partition subproblem can be solved by checking if there is an integer feasible solution to the following LP:

\[
\text{PARTLP:}\quad \min\quad 0 \\
\text{s.t.}\quad \sum_{j=1}^{i+C-1} x_j \geq 1 \quad 1 \leq i \leq n - C + 1 \tag{1}
\]

\[
\sum_{j=L(Q)+1}^{i+|Q|-1} x_j \leq k \quad \forall Q \tag{2}
\]

\[
x_1 = 1 \tag{3}
\]

\[
0 \leq x_j \leq 1 \quad 1 \leq j \leq n \tag{4}
\]

where \( x_i \) is a binary variable to denote if vertex \( i \) is the leftmost vertex of a block and \( L(Q) \) denotes the leftmost vertex of \( Q \). Constraints (3) ensure that some block must start at 1. Constraints (4) ensure that among \( C \)
consecutive vertices there must be at least one vertex which is the leftmost vertex of a block because a block has size at most $C$. Since a clique $Q$ intersects at most $k + 1$ blocks, constraints (2) ensure that the vertices in $Q$, except the leftmost, can include the leftmost vertices of at most $k$ blocks.

**Lemma 17.** If $x$ is a fractional solution to PARTLP then it can be rounded to an integer feasible solution $\bar{x}$ in polynomial time.

**Proof.** Consider the following rounding scheme which takes $O(n)$ time:

$$y_0 = 0, \quad y_i = \sum_{j=1}^{i} x_j \text{ and } \bar{x}_i = \begin{cases} 1 & \text{if } [y_{i-1}] \neq [y_i] \\ 0 & \text{otherwise} \end{cases} \text{ for all } 1 \leq i \leq n$$

Since each $\bar{x}_i$ is a 0-1 variable, $\bar{x}$ satisfies constraints in (1). Since $x_1 = 1$, by construction $\bar{x}_1 = 1$. Hence constraint (3) is also satisfied.

Now we prove by contradiction that $\bar{x}$ satisfies the constraints in (1). Suppose, the $i$th of such constraints is violated by $\bar{x}$. That implies all of $\bar{x}_1, \bar{x}_{i+1}, \ldots, \bar{x}_{i+C-1}$ are zero. Suppose $y_{i-1} = Z + \varepsilon$ where $Z$ is an integer and $0 < \varepsilon \leq 1$. Thus, $[y_{i-1}] = [Z + \varepsilon] = Z + 1$. Since, $\bar{x}_i, \bar{x}_{i+1}, \ldots, \bar{x}_{i+C-1}$ are all zero, $[y_{i+C-1}] = Z + 1$. Hence, $y_{i+C-1} \leq Z + 1$. Thus, $\sum_{j=i}^{i+C-1} x_j = y_{i+C-1} - y_{i-1} \leq 1 - \varepsilon < 1$ because $\varepsilon$ is strictly greater than 0. This implies that $x$ also violates the constraint. This is a contradiction.

Finally we prove that $\bar{x}$ satisfies constraints in (2) too. Again we prove by contradiction. Suppose $\bar{x}$ violates the constraint for clique $Q$. Then there are at least $k + 1$ non-zero $\bar{x}_i$s among those corresponding to the vertices in $Q$, except the leftmost vertex $i$. Let $i_1, i_2, \ldots, i_{k+1}$ be those vertices. Let $[y_{i_1}] = Z$. Since $\bar{x}_i$, is non-zero, $[y_{i_1-1}] = Z - 1$. Then $y_{i_1-1} \leq Z - 1$. The non-zero variables $\bar{x}_i$s imply that $[y_{k+1}] \geq Z + k$. That is, $y_{k+1} \geq Z + k - 1 + \varepsilon$ where $\varepsilon > 0$. Thus, $\sum_{j=i_1}^{i_1+|Q|-1} x_j \geq \sum_{j=i_1}^{i_{k+1}} x_j = y_{k+1} - y_{i_1} \geq Z + k - 1 + \varepsilon - Z + 1 = k + \varepsilon > k$. But this implies that $x$ also violates the constraint for $Q$. This is a contradiction. \hfill \square

### 4.5 A Combinatorial Algorithm

We now give a combinatorial algorithm for the partition subproblem. The algorithm does not use LP scaffolding and hence is more efficient. Towards achieving this, we find out the vertices which cannot be the rightmost vertex of a block in a valid partition. We call such vertices *forbidden*. The following two claims characterize the forbidden vertices.

**Claim 18.** Let $Q$ be a clique of size $kC + 2$ having vertices $i, i + 1, \ldots, i + kC + 1$. Then all the vertices $i + pC$ for $0 \leq p \leq k$ are forbidden.

**Proof.** Suppose there exists a block ending at a vertex $v = i + pC$. Consider the vertices $i, i + 1, \ldots, i + pC$. To cover these $pC + 1$ vertices, we need at least $p + 1$ blocks. The remaining $k - p$ blocks must cover the remaining $(k - p)C + 1$ vertices on the right of $v$. Since the size of block is at most $C$, this is not possible. \hfill \square

**Claim 19.** Let $Q$ be a clique of size $kC + 2 - r$ having vertices $i, i + 1, \ldots, i + kC + 1 - r$, where $1 \leq r \leq C - 1$. If the $r$ consecutive vertices $i + kC + 1 - r, i + kC + 1 - (r - 1), \ldots, i + kC$ are forbidden, then the vertices $i + pC$ for $0 \leq p \leq k - 1$ are also forbidden.

**Proof.** Since the vertices $i + kC + 1 - r, i + kC + 1 - (r - 1), \ldots, i + kC$ are forbidden, the block that covers these vertices must end at a vertex on the right of $i + kC$ and hence can cover at most $C - r - 1$ other vertices of $Q$. The remaining $k$ blocks must cover the remaining $(kC + 2 - r - 1) - (C - r - 1) = (k - 1)C + 2$ vertices. The same argument as in Claim 18 replacing $k$ by $k - 1$ gives us the forbidden vertices. \hfill \square

These observations lead to the algorithm COMBPARTITION for the partition subproblem.

**Lemma 20.** COMBPARTITION correctly solves the partition subproblem in $O(n + m)$ time.

**Proof.** The first step of the algorithm marks all possible forbidden vertices according to the rules of Claims 18 and 19. A single pass in the reverse canonical order is enough because a forbidden vertex can cause another vertex to be forbidden by the rule in Claim 19 only and the new forbidden vertex is always on the left of the original forbidden vertex.
Algorithm 2: \textsc{CombPartition}  

\begin{itemize}
    \item \textbf{Input}: A connected PIG $G = (V, E)$ with $\omega = kC + r$
    \item \textbf{Output}: If $G$ has a $[k + 1, C]$-partition; if \textsc{Yes} also output such a partition
    \begin{enumerate}
        \item For each vertex $v$ in the reverse canonical order, if there a clique with rightmost vertex $v$ that satisfy the conditions of Claims 18 and 19 mark the forbidden vertices;
        \item Starting with the leftmost vertex, greedily construct blocks from the vertices in canonical order as follows. Start a block with the leftmost uncovered vertex and extend till the rightmost non-forbidden vertex at a distance at most $C - 1$;
        \item In this process, if a block can not be constructed because of $C$ consecutive forbidden vertices, then output \textsc{No};
        \item Otherwise blocks are constructed covering all the vertices. Output \textsc{Yes};
    \end{enumerate}
\end{itemize}

\begin{figure}[h]
    \centering
    \includegraphics[width=\textwidth]{example.png}
    \caption{\textsc{CombPartition} is applied on two examples. Fig. (a) shows a PIG with $n = 8$, $C = 3$, $k = 1$, $kC + 2 = 5$, cliques $Q_1 = \{1, 2, 3, 4\}$, $Q_2 = \{2, 3, 4, 5\}$, $Q_3 = \{3, 4, 5, 6, 7\}$ and $Q_4 = \{4, 5, 6, 7, 8\}$. By Claim 18 for $Q_3$ and $Q_4$, vertices 3, 4, 6, 7 and 8 are forbidden. Since vertex 4 is forbidden, by Claim 19 for $Q_1$, vertex 1 is also forbidden. $Q_2$ does not create any new forbidden vertex because 5 is not forbidden. The algorithm creates first block $[1, 2]$ because 3 is forbidden. Then it creates two more blocks – $\{3, 4, 5\}$ and $\{6, 7, 8\}$. It is a valid $[k + 1 = 2, C = 3]$-partition. Fig. (b) shows a PIG with $n = 9$, $C = 2$, $k = 1$, $kC + 2 = 4$, cliques $Q_5 = \{1, 2, 3, 4\}$, $Q_6 = \{4, 5, 6\}$ and $Q_7 = \{6, 7, 8, 9\}$. By Claim 18 for $Q_5$ and $Q_7$, vertices 1, 3 and 6 are forbidden. Since 6 is forbidden, by the rule in Claim 19 for $Q_6$, vertex 4 is also forbidden. The algorithm outputs \textsc{No} because of the two consecutive forbidden vertices 3 and 4. So, there is no $[k + 1 = 2, C = 2]$-partition.
    
    \begin{itemize}
        \item If the algorithm outputs \textsc{No}, then there is a set of $C$ consecutive forbidden vertices. There can not be any valid partition because to cover these vertices we need a block of size at least $C + 1$. Hence the algorithm is correct.
        \item Now we prove that if the algorithm outputs \textsc{Yes}, the partition generated is a valid partition. Since the algorithm generates blocks of size at most $C$, the size constraint is satisfied. We only need to prove that no clique intersects more than $k + 1$ blocks generated by the algorithm. We prove this by contradiction.
        \item Suppose there is a clique $Q$ that intersects $k + 2$ blocks $B_0, B_1, \ldots , B_{k+1}$. Without loss of generality, we assume that only the leftmost vertex of $Q$ is covered by $B_0$ and only the rightmost vertex of $Q$ is covered by $B_{k+1}$. Because, otherwise we can take a sub-clique $Q' \subset Q$ with this property. Let the size of a block $B_j$ be $C - r_j$ for $1 \leq j \leq k$ where $0 \leq r_j \leq C$. Without loss of generality we assume that size of $B_0$ is $C$ and $r_0 = 0$. Also let $i$ be the leftmost vertex of $Q$.
        \item Size of $Q$ can not be $kC + 2$ because in that case $i$ would be forbidden but we assumed that $i$ is the rightmost vertex of $B_0$. Thus $|Q| \leq kC + 1$. An immediate corollary of Claim 21 is that each of $B_{j+p}$ has the rightmost vertex in between $i + (j + p - 1)C$ and $i + (j + p)C$, for $1 \leq p \leq k - j$. Hence, $B_k$ has its rightmost vertex in the segment $i + (k - 1)C + 1, \ldots , i + kC$. Hence $|Q| \geq 1 + (k - 1)C + 2 = (k - 1)C + 3$.
        \item So, $Q$ has size $1 + kC - \sum_{j=1}^{k} r_j + 1 = kC + 2 - r$ where $r = \sum_{j=1}^{k} r_j$, $1 \leq r \leq C - 1$ and by Claim 21 there are $r$ consecutive forbidden vertices on the immediate left of $i + kC + 1$. By Claim 19 $i$ is forbidden. It is a contradiction.
        \item Time complexity: The first step iterates over each vertex to check if there is a clique satisfying the conditions of Claims 18 and 19 and if so, marks the forbidden vertices. So there are two types of operations – (1) for checking the conditions and (2) marking the forbidden vertices. We can count the number of operations as}
\end{figure}
follows. For each of the first type of operations we can identify an edge between the leftmost vertex and the rightmost vertex of the clique being checked. For each marking operation, we can identify an edge from the leftmost vertex of the clique to the marked forbidden vertex. None of these edges is counted more than one operation. Hence total number of operations for the first step is \( O(n + m) \). The block generation step takes time \( O(n) \). Overall time complexity is \( O(n + m) \).

**Claim 21.** For each \( j \in \{0, 1, \ldots, k\} \) and for each \( t \in \{j, j + 1, \ldots, k\} \), all the \( \sum_{i=0}^{j} r_i \) consecutive vertices on the immediate left of \( i + tC + 1 \) are forbidden.

![Figure 5: An illustration of Claim 21](image)

**Proof.** We prove by induction on \( j \). Base step: \( j = 0 \). The statement is true because \( r_0 = 0 \) and \( i \) is not forbidden.

Induction step: \( j > 0 \). Let \( g_j = \sum_{i=0}^{j} r_i \). By induction hypothesis, for all \( j - 1 \leq t \leq k \), \( g_{j-1} \) consecutive vertices on the left of \( i + tC + 1 \) are forbidden. We just need to show that for \( j \leq t \leq k \), \( r_j \) consecutive vertices on the immediate left of \( i + tC + 1 - g_{j-1} \) are also forbidden. For this we show by induction on \( z \) that all the \( z \) consecutive vertices on the immediate left of \( i + tC + 1 - g_{j-1} \) are forbidden where \( 0 \leq z \leq r_j \).

For \( z = 0 \), the statement is true by the induction hypothesis on \( j \). For \( z > 0 \) we assume that all the \( z - 1 \) consecutive vertices on the immediate left of \( i + tC + 1 - g_{j-1} \) are forbidden. So it will be enough to prove that \( i + tC + 1 - g_{j-1} - z \) is also forbidden for all \( j \leq t \leq k \).

Let \( v \) be the rightmost vertex of \( B_{j-1} \). Then \( v = i + (j - 1)C - g_{j-1} \). Let \( w \) be the \( z \)th vertex on the left of \( v + C + 1 \). Then \( w = v + C + 1 - z = i + jC - g_{j-1} - z + 1 \). Since \( \text{COMB-partition} \) chose some vertex on the left of \( w \) as the rightmost vertex of \( B_j \), \( w \) must be forbidden.

We now ask the question – why is \( w \) forbidden? There must be a clique \( Q' \) of size \( kC + 2 - r' \), \( 1 \leq r' \leq C - 1 \) with the following properties – (i) \( w \) is its \( (qC + 1) \)th vertex, (ii) its leftmost vertex is \( u = i + (j - q)C - g_{j-1} - z + 1 \), (iii) starting from its rightmost vertex \( u + kC + 1 - r' \) to vertex \( u + kC \) are \( r' \) forbidden vertices, and (iv) the vertices \( u + pC \), \( 0 \leq p \leq k - 1 \) are forbidden. So we have the vertices \( u + pC = i + (j + p - q)C - g_{j-1} - z + 1 \) forbidden where \( 0 \leq p \leq k \). If we can prove that \( \{j, j + 1, \ldots, k\} \) is a subset of \( \{j - q, j - q + 1, \ldots, j - q + k\} \) then we are done. We see that \( j - q \leq j \). It will be enough if we show that \( j - q \geq 1 \). Because in that case \( j - q + k \geq k + 1 \).

Now consider the sub-clique \( Q'' \subset Q' \) of size \( |Q'| - (g_{j-1} + z - 1) = kC + 2 - (g_{j-1} + z + r' - 1) \) having the rightmost vertex same as the rightmost vertex of \( Q' \). The leftmost vertex of \( Q'' \) is \( i + (j - q)C \). Combining the induction hypotheses on \( j \) and \( z \) with property (iii) of \( Q' \), we see that there are \( g_{j-1} + z + r' - 1 \) forbidden vertices starting from its rightmost vertex \( u + kC + 1 - r' \) to vertex \( i + (j + k - q)C \). By Claim 19 all the vertices \( i + (j - q)C + pC, 0 \leq p \leq k \) are forbidden. Since we assumed that \( i \) is not forbidden, the leftmost of such forbidden vertices are on the right of \( i \), i.e., \( j - q \geq 1 \).

Combining the results of Lemmas 16 and 20 we get the proof of Theorem 1.

### 4.6 Algorithm for Splittable Weighted Problem

We consider a weighted PIG \( G = (V, E) \) with weight \( w_v \) on each vertex \( v \). WLOG, we assume \( w_v \leq C \) because number of output blocks is at least \( \sum W_v/C \) and we need at least those many output operations. Now we construct an unweighted PIG \( G' = (V', E') \) where \( V' \) has \( w_v \) copies of the vertex \( v \in V \) and there is an edge
in $E'$ between each copy of $u$ and each copy of $v$ for each edge $(u,v)$ in $E$. Let $|V| = n, |E| = m, |V'| = n' = \sum_{v} w_v, |E'| = m'$. The straight forward way of solving the splittable weighted problem on $G$ is to apply the algorithm for the unweighted problem on $G'$. This gives correct result but it makes the algorithm pseudo-polynomial as it takes $O(m' + n')$ time which is proportional to the sum of weights. This is mainly because both GreedyPartition and CombPartition iterate over each vertex in $G'$.

However, it turns out that iterating over each vertex in $G'$ is not necessary. It is easy to see that GreedyPartition can be modified to pack the blocks greedily with the vertices of $G$, or possibly with fractions of them, in time $O(n)$. We will see that for CombPartition too, it is enough to iterate over the maximal cliques in $G'$ in the reverse canonical order and mark all possible forbidden vertices in a maximal clique simultaneously. Even better, marking all forbidden vertices within a maximal clique are also not necessary. This is because forbidden vertices in each maximal clique $Q$ follow a particular pattern: starting from the leftmost vertex of $Q$, the first $t$ consecutive vertices of every $C$ vertices in $Q$ are forbidden. We say $t$ is the forbidden number of $Q$. We formally show this using the following two claims.

Claim 22. Let $Q$ be a maximal clique in $G'$ of size $kC + 2 - r$ having vertices $i, i + 1, \ldots, i + kC + 1 - r$, where $1 \leq r \leq C - 1$. If the $r + t$ consecutive vertices $i + kC + 1 - r, i + kC + 1 - (r - 1), \ldots, i + kC + t$ are forbidden, where $0 \leq t \leq C - 1$, then the vertices $i + pC + q$ for $0 \leq p \leq k - 1, 0 \leq q \leq t$ are also forbidden. Moreover, if $t$ is the largest possible then no more forbidden vertices are generated by the subcliques of $Q$ using Claims 18 and 19.

Proof. Consider the $t + 1$ cliques each of size $kC + 2 - r$ starting at $i, i + 1, \ldots, i + t$. Each of these cliques satisfy the conditions of Claim 18. Hence for each clique, starting from the leftmost vertex, every $C$th vertex is a forbidden vertex, i.e., all the vertices $i + pC + q$ for $0 \leq p \leq k - 1, 0 \leq q \leq t$ are forbidden. If $t$ is the largest then there are no other subclique which satisfy the conditions of Claims 18 and 19.

Claim 23. Let $Q$ be a maximal clique in $G'$ of size $kC + 2 + r$ having vertices $i, i + 1, \ldots, (i + kC + 1 + r)$, where $0 \leq r \leq C - 1$. If the $t$ consecutive vertices $j, j + 1, \ldots, (i + kC + r + t)$ are forbidden, where $0 \leq t \leq C - 1$, then the vertices $i + pC + q$ for $0 \leq p \leq k - 1, 0 \leq q \leq r + t$ are also forbidden. Moreover, if $t$ is the largest possible then no more forbidden vertices are generated by the subcliques of $Q$ using Claims 18 and 19.

Proof. Consider the $r + t + 1$ cliques each of size $kC + 2$ starting at $i, i + 1, \ldots, i + r + t$. Each of these cliques satisfy the conditions of Claim 19. Hence for each clique, starting from the leftmost vertex, every $C$th vertex is a forbidden vertex, i.e., all the vertices $i + pC + q$ for $0 \leq p \leq k - 1, 0 \leq q \leq r + t$ are forbidden. If $t$ is the largest then there are no other subclique which satisfy the conditions of Claim 19 and even if there is a subclique which satisfy the conditions of Claim 18 it will generate forbidden vertices which are already considered.

So we just need to store the forbidden numbers of the maximal cliques in $G'$. Since only the maximal cliques of size larger than $(k - 1)C + 2$ may possibly generate new forbidden vertices by Claims 22 and 23, we can be determined using the forbidden numbers of the maximal cliques they intersect. The blocks of the required partition, as determined by the second step of CombPartition, can also be determined using the forbidden numbers of the maximal cliques.

Lemma 24. There exists an implementation of algorithm CombPartition which takes time $O(m + n)$ for the unweighted PIG $G'$.

Proof. To simplify the computation of forbidden numbers of the maximal cliques, we maintain the information about the forbidden vertices discovered so far, as a sequence of intervals sorted on their left endpoints. Each interval represents a maximal block of consecutive forbidden vertices. For each maximal clique $Q$, we find out the interval which intersect the rightmost vertex $z$ of $Q$ in $O(k)$ time and compute the largest number of consecutive forbidden vertices on the right of $z$ including itself. We then compute the forbidden number of $Q$ using the rules in Claims 22 and 23 as applicable. Each of these two computations takes constant time. A non-zero forbidden number implies a set of $k + 1$ new intervals. We merge this new set of intervals to the sequence of intervals in $O(k)$ time. If two intervals intersect, they are replaced by their union. If at any point the size of an interval becomes at least $C$, we stop. There can not be any $(k + 1, C]$-partition. If there are $q$ maximal cliques, this process completes in time $O(qk)$ and there are $O(qk)$ intervals in the sequence.

Starting from the left most interval, we decide the size of the next block (as determined in the second step of CombPartition) and fully pack the block with the vertices of $G$ in a greedy manner, possibly taking only
a fraction of a vertex, if required. Over all, it takes time $O(n + qk)$ because in each step either one interval is used up or a vertex of $G$ is completely used up (note that each vertex in $G$ has weight at most $C$).

It takes $O(n + m)$ time to find the maximal cliques. Thus total time taken is $O(n + qk) = O(n + m)$, as we will show that $qk = O(m)$. Since each vertex in $G$ has weight at most $C$, the number of vertices in a maximal clique in $G$ is at least $((k - 1)C + 2)/C = \Omega(k)$. The number of edges in $G$ from the leftmost vertices of all maximal cliques is $\Omega(qk)$. Thus $m \geq \Omega(qk)$ implying that $qk = O(m)$.

Combining the results of Lemmas 25 and 24 we get the proof of Theorem 2.

5 Partition on Split Graphs

Since split graphs are also chordal, solving the partition (not block-partition) problem is enough. It can be noted that the same lower bound of Lemma 14 applies here too.

5.1 Upper Bound

**Lemma 25.** Let $\omega$ be the clique number of a split graph $G$. There exists a polynomial time algorithm that gives a $[[\omega/C] + 1, C]$-partition for $G$.

**Proof.** Let the vertex set of $G$ be split into clique $Q$ and independent set $S$. WLOG we assume that $Q$ is a maximum clique. Because otherwise $|Q| = \omega - 1$ and we can move a vertex in $S$ that is adjacent to all vertices in $Q$ to $Q$. Now consider the following partition of vertices: $\Pi = \{P_1, P_2, \ldots, P_t\} \cup \{v\} | v \in S \}$ where $t = [\omega/C]$, and $\{P_i\}_{i=1}^t$ is an arbitrary partition of $Q$ such that $|P_i| = C$ for all $i = 1, \ldots, (t - 1)$. Note that this partition can be created in polynomial time. Each part is connected and has at most $C$ vertices. Moreover, any maximal clique in $G$ intersects at most $t + 1$ parts. Thus, $\Pi$ is a $[[\omega/C] + 1, C]$-partition.

5.2 NP-hardness

Since the upper bound and the lower bound differ by 1, it is enough to decide if $G$ has a $[[\omega/C], C]$-partition or not. If the answer is Yes then we have an optimal solution to the partition problem with clique intersection $\lambda = [\omega/C]$. Otherwise the partition given in the proof of Lemma 25 gives an optimal solution with clique intersection $\lambda = [\omega/C] + 1$. Thus Lemma 26 directly gives a proof of Theorem 3.

**Lemma 26.** The problem of deciding if a split graph $G$ has a $[[\omega(G)/C], C]$-partition for $C \geq 2$ is $NP$-complete.

**Proof.** We show that the decision problem is $NP$-complete even for $C = 2$. We call the problem in this special case as CP. First we show that CP is in $NP$. A maximal clique in $G$ is either $Q$ or the closed neighborhood of a vertex in $S$. So the maximal cliques in $G$ can be found in polynomial time. Suppose a partition of the vertices is given. Size constraints can be easily checked. Each part contains a single vertex or a pair of vertices. A single vertex is trivially connected. Connectedness of a part of size 2 can be checked by just checking if there is an edge between the two vertices. Clique intersection constraint can also be checked in polynomial time.

We now introduce a set partitioning problem (SP) is defined as follows. Given a set of $2n$ elements $e_1, e_2, \ldots, e_{2n}$ and a collection of $m$ subsets $S_1, S_2, \ldots, S_m$, can the elements be partitioned into $n$ groups of size 2 such that each subset has both elements of at least one group?

We complete the proof by first showing a polynomial time reduction from SP to CP (Claim 27) and then a polynomial time reduction from the well known NP-complete problem SAT to SP (Claim 28).

**Claim 27.** $SP \leq_P CP$.

**Proof.** Given an instance of SP, we construct an instance of CP as follows. The complete set $Q$ has a vertex $v_i$ corresponding to each element $e_i$ and the independent set $S$ has a vertex $w_j$ corresponding to each subset $S_j$. There is an edge between $v_i$ and $w_j$ if and only if $e_i \notin S_j$. Clearly $\omega = 2n$ and hence $[\omega/C] = n$.

Suppose there is a Yes solution to the SP instance where the groups are $G_1, G_2, \ldots, G_n$. Then create a partition $\Pi = \{P_1, P_2, \ldots, P_n\} \cup \{w_j\}_{j=1}^m$ for the CP instance where $v_k \in P_i$ if and only if $e_k \in G_i$. Clearly each part is connected and has at most 2 vertices. Clique intersection constraint is satisfied for $Q$. Since $S_j$ contains both elements of at least one $G_i$, the maximal clique $Q'$ containing $w_j$ does not intersect at least one part $P_i$. Including the part $\{w_j\}, Q'$ intersects at most $(n - 1) + 1 = n$ parts. Hence $\Pi$ is a $[n, 2]$-partition.
On the other hand, suppose there is a Yes solution to the CP instance. Since the clique intersection constraint is satisfied for \( Q \), the vertices of \( Q \) are divided into parts of size exactly 2. These parts give the required groups of SP because, for a maximal clique \( Q' \) containing \( w_j \) has clique intersection at most \( n \), and hence it must not intersect with at least one part \( P_i \) which implies that \( S_j \) contains both elements of \( G_i \).

\[ \text{Claim 28.} \ SAT \leq_P \ SP. \]

\[ \text{Proof.} \] Suppose an instance of SAT has \( p \) Boolean variables \( x_1, x_2, \ldots, x_p \) and \( q \) clauses \( C_1, C_2, \ldots, C_q \). WLOG we assume that there is at most one literal for each variable in each clause. Now we construct an instance of SP as follows. There are \( 4p \) elements \( x_1, x'_1, T_1, F_1, x_2, x'_2, T_2, F_2, \ldots, x_p, x'_p, T_p, F_p \) and the subsets are of two types as follows: (1) the subsets \( \{x_i, x'_i, T_i, F_i\}, \{x_i, x'_i, F_i\}, \{x_i, T_i, F_i\}, \{x'_i, T_i, F_i\} \) for all \( 1 \leq i \leq p \), and (2) the subset \( \cup_{t \in C_i} \{l_t, T_i\} \) for all clause \( C_j \), where \( l_t \) is either \( x_i \) or \( x'_i \) (e.g., for \( C_j = (x_1 + x_2 + x_3) \) the subset \( \{x_1, T_1, x'_2, T_2, x_3, T_3\} \)).

Suppose there is a satisfying assignment for the SAT instance. Then construct a grouping for the SP instance as follows. For all \( i \), if \( x_i \) is true then construct two groups \( \{x_i, T_i\} \) and \( \{x'_i, F_i\} \); otherwise (i.e., \( x_i \) is false) construct two groups \( \{x_i, F_i\} \) and \( \{x'_i, T_i\} \). Clearly each subset of type (1) has two elements belonging to the same group. Since each clause is satisfied there must be a variable \( x_i \) such that one of \( x_i \) and \( x'_i \) is true. Hence the corresponding subset of type (2) must have two elements belonging to the same group.

On the other hand, suppose there is Yes solution for the SP instance. The subsets of type (1) force the elements \( x_i, x'_i, T_i, F_i \) to form 2 groups amongst themselves. The subsets of type (2) ensure that one of the literals \( l_t \) in the clause \( C_j \) must group with \( T_i \) and hence \( C_j \) must be true. This implies that the SAT instance has an satisfying assignment implied by the groups. For some \( x_i \) the grouping may contain \( \{x_i, x'_i\}, \{T_i, F_i\} \), in which case the value for the variable \( x_i \) can be chosen arbitrarily.

\[ \Box \]

6 Conclusions and Future Work

We gave polynomial time algorithms for unweighted and splittable weighted versions of the component coloring problem for proper interval graphs and showed that it is NP-hard for split graphs. However the complexity of both the versions are not known for general interval graphs. We would like to get polynomial time algorithms for general interval graphs using similar ideas. This may lead to a constant factor approximation algorithm for the weighted version of the problem for general interval graphs which is known to be NP-hard, using ideas from Bin-packing.

References


