

VColor: A Practical Vertex-cut Based Approach for Coloring Large Graphs

Yun Peng^{*}, Byron Choi[†], Bingsheng He[‡], Shuigeng Zhou[§], Ruzhi Xu^{*} and Xiaohui Yu[¶]

^{*}Research Center of Big Data Application, Qilu University of Technology

Email: csypeng, xrz@foxmail.com

[†]Department of Computer Science, Hong Kong Baptist University

Email: bchoi@comp.hkbu.edu.hk

[‡]School of Computer Engineering, Nanyang Technological University

Email: bshe@ntu.edu.sg

[§] SKLIIP and School of Computer Science, Fudan University

Email: sgzhou@fudan.edu.cn

[¶]School of Computer Science, Shandong University

Email: xyu@sdu.edu.cn

Abstract—Graph coloring is a fundamental NP-hard problem in graph theory. It has a wide range of real applications, such as Operations Research, Communication Network, Computational Biology and Compiler Optimization. Notable efforts have been spent on designing its approximation algorithms. Halldrsson proposed the algorithm (denoted as `SampleIS`) with the current best known approximation ratio. However, its time complexity is $O(|G|^3)$, where $|G|$ is the number of vertices of a graph G . It is clear that `SampleIS` is not practical for large graphs. In this paper, we propose a practical vertex-cut based coloring technique (`VColor`) for coloring large graphs. First, we partition G into k connected components (CCs) of a small size s by removing a vertex-cut component (VCC). For each CC, we apply our novel coloring algorithm, based on maximal independent set enumeration. The approximation ratio and the time complexity for coloring the k CCs are $\log s + 1$ and $O(ks^2 3^{s/3})$, respectively, whereas those of `SampleIS` are $ks(\log \log ks)^2 / \log^3 ks$ and $O(k^3 s^3)$. For the VCC, we simply apply `SampleIS`. To combine the colorings of the CCs and the VCC, we propose a maximum matching based algorithm. Second, in the context of a database of graphs, users may color many graphs. We propose an optimization technique, inspired by multi-query optimization, for coloring a set of graphs. We design a VP hierarchy (VPH) to represent the common subgraphs as the common CCs. Third, we propose techniques for determining the optimal values of the parameters of `VColor`. Our extensive experimental evaluation on real-world graphs confirms the efficiency and/or effectiveness of our proposed techniques. In particular, `VColor` is more than 500 times faster than `SampleIS`, and the number of colors used are comparable on real graphs Yeast and LS.

I. INTRODUCTION

This paper revisits the classical *graph coloring problem*. It is to color the vertices of a graph using the fewest colors such that no two adjacent vertices having the same color. For example, Fig. 1 presents two graphs and their colorings.

The importance of graph coloring has long been recognized in the literatures of Operations Research, Communication Network, Computational Biology and Compiler Optimization, among others:

- *Nucleic Acid Sequence Design*. Given a set of nucleic acids, a dependency graph [1] is a graph, where each

vertex is a nucleotide and two vertices have an edge iff the two nucleotides form a base pair in at least one of the nucleic acids. A coloring of the dependency graph is a nucleic acid sequence that is compatible with the set of nucleic acids.

- *Frequency Assignment*. A cellular phone network is modeled as a graph, where a vertex is a base station and two vertices are neighbor iff the two base stations are in communication range. When assigning frequencies to the base stations, the neighbors that are close to each other need to be assigned to different frequencies to avoid interference. It is exactly a graph coloring of the network [2].
- *Compiler Optimization*. The register allocators of almost all modern production compilers are based on graph coloring [3]. Specifically, given a set of registers and values, one may construct an interference graph, where a vertex is the live range of a value and an edge indicates that the two live ranges have overlappings and interference with each other. Register allocation is equivalent to color the interference graph.
- *Scheduling*. Assume that we have to schedule a set of interfering jobs (e.g., scheduling aircrafts to flights). We can construct a conflict graph, where the vertices are jobs and two vertices have an edge iff the corresponding jobs cannot be executed at the same time. Let colors denote available time slots and each job needs a time slot. The coloring of the conflict graph with minimum number of colors is the schedule of the smallest time span [4].

Since graph coloring is NP-hard, notable efforts have been spent to propose approximation algorithms (e.g., [5], [6], [7], [8], [9], [10]). However, there are at least three technical challenges to directly apply them to color the graphs nowadays. (I) Firstly, Halldrsson [6] have proposed the approximation algorithm (denoted as `SampleIS`) with the current best known approximation ratio $|G|(\log \log |G|)^2 / \log^3 |G|$. However, its time complexity is $O(|G|^3)$. It is hence impractical to cope

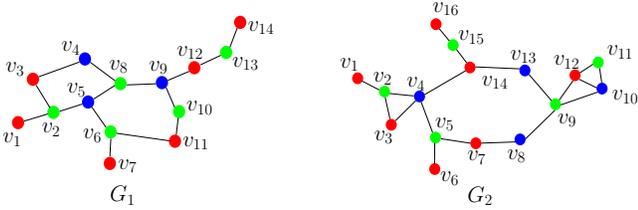


Fig. 1: An example of two graphs and their colorings

with the scale of real graphs nowadays. For example, in our preliminary experiments, `SampleIS` takes 610 and 2,381 seconds for coloring a biology graph `Yeast` and a latin square graph `LS` with just 3K and 0.9K vertices, respectively, and does not finish after running on a road network (having 260K vertices) for one week. (II) Secondly, in practice, a graph database may contain numerous of graphs. For example, the popular biological graph database `PubChem` contains millions of graphs [11]. Real graphs often have many common substructures (e.g., as reported in [12], [13]). In `Nucleic Acid Sequence Design`, the biologists often study several sequences and hence they will color multiple dependency graphs. The dependency graphs often have common substructures as the number of distinct base pairs is small in practice. In `Complier Optimization`, coloring many interference graphs is needed because a computer often runs several programs. The interference graphs have common substructures as the programs invokes common libraries. However, existing methods mainly focus on the coloring of a single graph, where the common substructures among the graphs are not well utilized. There are hence many optimization opportunities when compared to directly applying existing methods to color each graph individually. (III) Thirdly, many graph coloring methods involve a few parameters and their values affect the running times.

In this paper, we propose a novel vertex-cut based coloring framework (`VColor`) to address the challenges. For challenge (I), we propose to color a graph G in a *divide and conquer* manner. Specifically, we partition G to k *connected components* (CCs) of a small size s by removing a *vertex-cut component* (VCC). The advantage of this vertex-cut partition is that there is no crossing edge between the CCs, s.t. each CC can be colored independently. To color the CCs, we design a coloring algorithm based on maximal independent set enumeration. Its approximation ratio is $\log s + 1$ and time complexity is $O(ks^2 3^{s/3})$. These are much better than the approximation ratio $ks(\log \log ks)^2 / \log^3 ks$ and time complexity $O(k^3 s^3)$ of `SampleIS` on the CCs. To color the vertex-cut component VCC, we simply adopt `SampleIS`, because VCC can often be small enough such that $O(|VCC|^3)$ is often affordable. To combine the local colorings, we propose an optimal algorithm based on the maximum matching between the local colorings. *Our experiments show the our technique can significantly reduce the running time, while keeping the number of colors comparable to SampleIS.* In particular, in our experiments, our technique takes only 1.2 and 3.2 seconds to color `Yeast` and `LS`, which are about 500 and 700 times faster than `SampleIS`, respectively, and the numbers of colors are comparable to those of `SampleIS`. The reasons of using comparable number of colors with `SampleIS` are as follows. i) `VColor` and `SampleIS` both essentially iteratively extract large independent sets (ISs). ii) The size of the IS found by `VColor` and that found by `SampleIS` in each iteration are relatively close.

For challenge (II), we propose a technique, inspired by multi-query optimization, to optimize the coloring of a set of graphs. Since `VColor` colors graphs based on their vertex-cut partitions (VPs), we can take the common subgraphs as the common CCs of the VPs of the graphs. In this way, the common subgraphs are colored just once and the overall computation is hence reduced. We propose a vertex-cut partition hierarchy (VPH) to represent the common CCs of the VPs of the graphs. Since constructing an optimum VPH to minimize the coloring time is NP-hard, a heuristic algorithm is proposed to construct an optimal one. Our technique can significantly reduce the running time to color a set of graphs. For example, on a set of 800 graphs, our technique is about 2 times faster than coloring each graph individually.

For challenge (III), we propose a cost model for the coloring time of a VPH in terms of the parameters of our technique. We propose an efficient sampling based method to estimate the optimal values of the parameters with the cost model. Our experiments show that on `Pokec` and `PA` graphs, given a set of possible values of the parameters, the VPH using the values that are estimated optimal by the cost model uses only at most 15% and 18% more time than the VPH using the optimum values.

In summary, the contributions of this paper are as follows.

- We propose a vertex-cut based approach `VColor` for coloring large graphs. The coloring results are comparable with those of `SampleIS`, which is the algorithm with the best known approximation ratio. Further, we are two orders of magnitudes faster on `Yeast` and `LL` and produce coloring results even when `SampleIS` cannot finish.
- We propose a technique to optimize the coloring of a set of graphs. A vertex-cut partition hierarchy (VPH) is designed to represent the common CCs of the VPs of the graphs. A heuristic algorithm is proposed to construct an optimal VPH.
- We propose a cost model to express the coloring time of our technique in terms of certain parameters. An efficient sampling based search method is proposed to estimate the values of the parameters that minimize the running time on a given set of graphs.
- Our experiments verify the effectiveness and efficiency of our techniques on real-world graphs.

Organizations. The rest of this paper is organized as follows. Sec. II provides the background of this paper. Sec. III proposes the vertex-cut based coloring technique. The techniques for optimizing the coloring of a set of graphs is detailed in Sec. IV. The experimental evaluations are reported in Sec. V. Sec. VI discusses the related work and Sec. VII concludes this paper.

II. PRELIMINARIES AND PROBLEM DEFINITION

A. Notations on graphs

We start by recalling some notations for graphs. This paper studies *undirected graphs*, or simply *graphs*. A graph is denoted as $G = (V, E)$, where $V(G)$ and $E(G)$ are the vertex set and the edge set of G , respectively. $|G|$ denotes the size of

TABLE I: Notation table

$\mathcal{P}, \mathcal{P}_G$	vertex-cut partition of G
$VCC(\mathcal{P})$	vertex-cut component in \mathcal{P}
$CC(\mathcal{P})$	connected component in \mathcal{P}
s	size of connected component in \mathcal{P}
VPB	vertex-cut partition bigraph
\mathcal{H}	VP hierarchy
L	the number of VPBs in \mathcal{H}
Δ, Δ_G	the largest degree of the vertices of G
α	number of colors

G and $|G| = |V(G)|$. $N(v)$ and $N(S)$ denote the neighbors of $v \in V(G)$ and $S \subseteq V(G)$, respectively. $\bar{N}(v)$ and $\bar{N}(S)$ denote the non-neighbors of v and S , respectively. Δ denotes the largest degree of vertices in G . A *vertex-cut* of G is a set of vertices of G whose removal makes G disconnected. An *independent set (IS)* I of G is a subset of $V(G)$, such that $\forall u, v \in I, (u, v) \notin E(G)$. A *maximal independent set (MIS)* M of G is an IS, such that $M \cup \{v\}$ is not an IS, for any $v \in V(G) \setminus M$. $\mathcal{M}(G)$ denotes the set of all MISs of G . An *independent set partition* \mathcal{I} of G is a set of nonempty subsets of $V(G)$, where $\forall I \in \mathcal{I}$ is an IS of G , $\forall I, I' \in \mathcal{I}, I \cap I' = \emptyset$ and $I \cup I'$ is not an IS of G and $\cup_{I \in \mathcal{I}} I = V(G)$. The size of an IS partition \mathcal{I} is the number of ISs in it.

Definition 1: A coloring of G is an assignment of a unique color to a vertex of G such that no two neighboring vertices are assigned the same color.

If a graph G can be colored using α colors, G is called α -colorable. The minimum value of α is called the *chromatic number* of G , denoted by χ_G . The set of vertices assigned with the same color is called a *color class*.

Proposition 1: An α -coloring of G is equivalent to an IS partition of G of size α , where each IS is a color class.

Problem definition. Given a set of graphs \mathcal{D} , color each graph G in \mathcal{D} with as few colors as possible.

Graph coloring is NP-complete and we study its optimization problem in this paper.

B. SampleIS Algorithm

In this subsection, we review the SampleIS algorithm [6], which is the approximation algorithm with the best known approximation ratio. It is presented in Fig. 2. The main idea of SampleIS is to iteratively extract a large independent set (IS) from G until G is empty. SampleIS has two sub-functions SampleIS_step and CliqueRemoval to compute a large IS of a graph G . The larger IS computed by SampleIS_step and CliqueRemoval is extracted in each iteration.

The rationale of SampleIS_step is as follows. Given an α -colorable graph G , SampleIS_step samples an IS I of G of size $\log_\alpha |G|$. Then, it recursively processes $\bar{N}(I)$. For a higher probability of sampling a large IS in $\bar{N}(I)$, $\bar{N}(I)$ should be large. The recursion stops until $\bar{N}(I)$ is small enough and it invokes CliqueRemoval to compute an IS, as CliqueRemoval guarantees to find a large IS from a small $\bar{N}(I)$. SampleIS_step assumes G is α -colorable. However, the value of α is not known in advance. Therefore, SampleIS tries all values of $\alpha \in [1, \Delta + 1]$ (Line 02), as a graph is assured colorable by $\Delta + 1$ colors [14].

Before we describe CliqueRemoval, we first describe its sub-function Ramsey. The rationale of Ramsey is to choose

```

Procedure SampleIS
Input: A graph  $G$ 
Output: A minimal graph coloring of  $G$ 
01 while  $|G| > 0$ 
02    $I_1 = \max_{\alpha=1}^{\Delta+1} \text{SampleIS\_step}(G, \alpha)$ 
03    $I_2 = \text{CliqueRemoval}(G)$ 
04   removing  $\max(I_1, I_2)$  from  $G$ 

function SampleIS_step
05 if  $|G| \leq 1$  return  $G$ 
06 while true
07   randomly pick a set  $I$  of  $\log_\alpha |G|$  vertices from  $G$ 
08   if  $I$  is an independent set
09     if  $|\bar{N}(I)| \geq \frac{|G| \log |G|}{2\alpha \log \log |G|} - |I|$ , return  $I \cup \text{SampleIS}(\bar{N}(I))$ 
10   else
11      $I' = \text{CliqueRemoval}(\bar{N}(I)) \cup I$ 
12     if  $|I'| \geq \frac{\log^3 |G|}{6 \log \log |G|}$ , return  $I'$ 

function CliqueRemoval
13  $i = 0$ 
14 while  $|G| > 0$ 
15    $(C_i, I_i) = \text{Ramsey}(G)$ 
16   remove  $C_i$  from  $G$ 
17    $i = i + 1$ 
18 return  $\max_i I_i$ 

function Ramsey
19 if  $|G| = 0$  return  $(\emptyset, \emptyset)$ 
20 choose some  $v \in G$ 
21  $(C_1, I_1) = \text{Ramsey}(N(v))$ 
22  $(C_2, I_2) = \text{Ramsey}(\bar{N}(v))$ 
23 return  $(\max(C_1 \cup \{v\}, C_2), \max(I_1, I_2 \cup \{v\}))$ 

```

Fig. 2: Procedure SampleIS

a pivot v and recursively process both the non-neighbors and neighbors of v . It examines the neighbors of v as the pivot v may be a bad choice and its neighbors may have a large IS. The same logic is also used to compute a large clique. Ramsey has a property that if the clique found is large, the IS found is small; otherwise, the IS found is large. Hence, CliqueRemoval iteratively applies Ramsey and removes the clique from G . The largest IS found in all iterations is returned.

The approximation ratio of SampleIS is $|G|(\log \log |G|)^2 / \log^3 |G|$, which is the best known approximation ratio. However, the time complexity of SampleIS is as high as $O(|G|^3)$, as the time complexity of SampleIS_step is $O(|G|^2)$ and the while loop (Lines 01-04) may execute $O(|G|)$ times. SampleIS is hence inefficient on large graphs in practice.

III. VERTEX-CUT BASED GRAPH COLORING

In this section, we propose our vertex-cut based graph coloring technique (VColor). Our main steps are the following: (i) partitioning the input graph G to a set of connected components (CCs) by removing a vertex-cut component (VCC); (ii) coloring the CCs and the VCC separately; and (iii) combining the local colorings. The advantage is that VColor can significantly reduce the running time while keeping the coloring results comparable with that of SampleIS. We focus on coloring one graph in this section and study coloring of a set of graphs in the next section.

A. Vertex-cut partition

We first define the vertex-cut partition used in VColor.

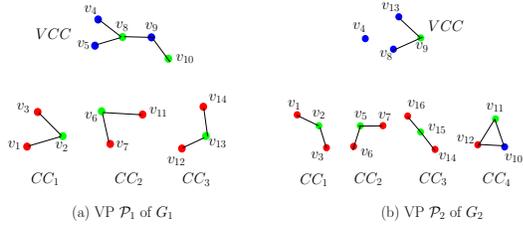


Fig. 3: An example of vertex-cut partitions

Definition 2: Given a graph G and a parameter s , a *Vertex-cut Partition* (VP) of G is a graph partition $\mathcal{P} = \{CC_1, CC_2, \dots, CC_{\lfloor \frac{|G| - |VCC|}{s} \rfloor}, VCC\}$, where VCC is the vertex-cut component, removing which leads to connected components $\{CC_1, \dots, CC_{\lfloor \frac{|G| - |VCC|}{s} \rfloor}\}$ of the same size s .

Note that the number of CC 's in \mathcal{P} can be $\lceil \frac{|G| - |VCC|}{s} \rceil$ and there may often exist CC 's that are smaller than s , but it is ignored for presentation simplicity. We use $VCC(\mathcal{P})$ to denote the VCC of \mathcal{P} and $CC(\mathcal{P})$ to denote the CC 's $\{CC_1, CC_2, \dots, CC_{\lfloor \frac{|G| - |VCC|}{s} \rfloor}\}$ of \mathcal{P} .

Example 1: Suppose $s = 3$, Fig. 3 presents the vertex-cut partitions of G_1 and G_2 that are shown in Fig. 1.

The vertex-cut partition has two properties as follows.

Property 3.1: Given a VP \mathcal{P} of a graph G , suppose CC_i in \mathcal{P} can be colored using α_i colors, $CC(\mathcal{P})$ can be colored using $\max\{\alpha_1, \alpha_2, \dots, \alpha_{|CC(\mathcal{P})|}\}$ colors.

Proof: Since there is no crossing edge between the CC 's in \mathcal{P} , any vertex $v_i \in CC_i$ can have the same color with any vertex $v_j \in CC_j$, for $i \neq j$. Without loss of generality, suppose $\alpha_j = \max\{\alpha_1, \alpha_2, \dots, \alpha_{|CC(\mathcal{P})|}\}$. For any $CC_i \neq CC_j$, we can any pick α_i colors from the α_j colors of CC_j to color CC_i . It can guarantee that no two neighboring vertices are assigned with the same color. Therefore, all CC 's in \mathcal{P} can be colored using $\max\{\alpha_1, \alpha_2, \dots, \alpha_{|CC(\mathcal{P})|}\}$ colors. ■

Property 3.2: Given a VP \mathcal{P} of a graph G , suppose $CC(\mathcal{P})$ can be colored using α_{cc} colors and VCC can be colored using α_{vcc} colors, G can be colored using at most $\alpha_{cc} + \alpha_{vcc}$ colors.

Proof: Note that there can be crossing edges between the CC 's and the VCC . Therefore, if we color the CC 's and the VCC with different colors, *i.e.*, no color of CC 's is used to color VCC , we can guarantee that no two neighboring vertices are assigned with the same color. Therefore, G can be colored using at most $\alpha_{cc} + \alpha_{vcc}$ colors. ■

Note that $\alpha_{cc} + \alpha_{vcc}$ is just an upper bound of the number of colors to color G . Our coloring algorithm proposed in the following section can combine the colorings of $CC(\mathcal{P})$ and the $VCC(\mathcal{P})$ and return a minimal coloring of G .

B. Coloring algorithm

By exploiting Properties 3.1 and 3.2, we propose our graph coloring algorithm as shown in Fig. 4. Given a graph G and a VP \mathcal{P} of G , Line 01 colors the VCC in \mathcal{P} using `SampleIS`. For each CC_i in \mathcal{P} , Line 03 computes the set of all MISs of CC_i , $\mathcal{M}(CC_i)$, using MIS enumeration algorithms (*e.g.*, [15], [16]). Line 04 colors CC_i based on selecting from $\mathcal{M}(CC_i)$ a minimal IS partition of CC_i that can cover CC_i . Finally, the colorings of the CC 's and the VCC are combined (Line 05).

```

Procedure Color
Input: A graph  $G$ , a VP  $\mathcal{P}$  of  $G$ 
Output: A minimal IS partition of  $G$ 
01  $\mathcal{I}_{vcc} = \text{SampleIS}(VCC)$ 
02 for each  $CC_i$  in  $\mathcal{P}$ 
03    $\mathcal{M}(CC_i) = \text{MISEnum}(CC_i)$  //enumerate all MISs of  $CC_i$ 
04    $\mathcal{I}_i = \text{ISPartition}(\mathcal{M}(CC_i), CC_i)$ 
05 return  $\text{comb}(\mathcal{I}_{vcc}, \mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_k, G)$ 

function ISPartition( $\mathcal{M}, G$ )
06  $\mathcal{I} = \emptyset$ 
07 while  $|\cup_{I \in \mathcal{I}} I| < |G|$ 
08    $M = \arg \max_{M \in \mathcal{M}} |\cup_{I \in \mathcal{I}} I \cup \{M\}| - |\cup_{I \in \mathcal{I}} I|$ 
09    $I = M - \cup_{I \in \mathcal{I}} I$ 
10   add  $I$  to  $\mathcal{I}$ 
11 return  $\mathcal{I}$ 

function comb( $\mathcal{I}_{vcc}, \mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_k, G$ )
12 for each  $i = 1$  to  $k$ 
13   construct an empty bigraph  $B$ 
14   for each  $I \in \mathcal{I}_{vcc}$ , insert a vertex to  $B$ 
15   for each  $J \in \mathcal{I}_i$ , insert a vertex to  $B$ 
16   for each  $I \in \mathcal{I}_{vcc}$  and  $I' \in \mathcal{I}_i$ 
17     if  $N(I) \cap I' = \emptyset$ 
18       insert an edge  $(I, I')$  to  $B$ 
19   compute a maximum matching  $M$  of  $B$ 
20   for each  $I' \in \mathcal{I}_i$ 
21     if  $\exists I \in \mathcal{I}_{vcc}$  s.t.  $(I, I') \in M$ 
22        $I = I \cup I'$ 
23     else add  $I'$  to  $\mathcal{I}_{vcc}$ 
24 return  $\mathcal{I}_{vcc}$ 

```

Fig. 4: Procedure Color

Function `ISPartition` computes a minimal IS partition of CC_i based on the heuristic of `SetCover` (Lines 06-11). In each iteration, it selects the MIS M in \mathcal{M} that can maximize the marginal increase of the number of covered vertices (Line 08). Line 09 computes a disjoint IS I by removing the vertices already covered from M . Line 10 adds I to the IS partition.

Function `comb` combines the IS partitions to obtain the final minimal IS partition (*i.e.*, a coloring). Note that $(\cup_{i=1}^k \mathcal{I}_i) \cup \mathcal{I}_{vcc}$ is an IS partition of G , but it may not be minimal. Thus, Function `comb` (Lines 12-24) merges the ISs in $(\cup_{i=1}^k \mathcal{I}_i) \cup \mathcal{I}_{vcc}$ whose union is still an IS of G in order to obtain a smaller IS partition of G . Since $\forall I \in \mathcal{I}_i$ and $\forall I' \in \mathcal{I}_j$, $i \neq j$, $I \cup I'$ is an IS of G , we just determine the combination of \mathcal{I}_{vcc} with each \mathcal{I}_i , respectively (Lines 13-23). The minimum IS partition \mathcal{I} combined from \mathcal{I}_{vcc} and \mathcal{I}_i is an IS partition such that I is a subset of some IS in \mathcal{I} , for any $I \in \mathcal{I}_{vcc} \cup \mathcal{I}_i$. We find that it is equivalent to compute the maximum matching of a bigraph $B = (\mathcal{I}_{vcc} \cup \mathcal{I}_i, E)$, where $(I, I') \in E$ iff I has no neighbor in I' . Therefore, Lines 13-18 construct such a bigraph, Line 19 computes the maximum matching and Lines 20-23 compute the unions of the ISs that are in the maximum matching.

Example 2: Let us color the graph G_1 in Fig. 1 using the vertex-cut partition shown in Fig. 3. Proc. `Color` first colors VCC using `SampleIS`, and obtains a minimal IS partition $\mathcal{I}_{vcc} = \{\{v_4, v_5, v_9\}, \{v_8, v_{10}\}\}$. Then, we enumerate all MISs and compute a minimal IS partition using `SetCover` for each CC . $\mathcal{I}_1 = \{\{v_1, v_3\}, \{v_2\}\}$, $\mathcal{I}_2 = \{\{v_7, v_{11}\}, \{v_6\}\}$ and $\mathcal{I}_3 = \{\{v_{12}, v_{14}\}, \{v_{13}\}\}$. Finally, we combine the local colorings. After combining \mathcal{I}_1 with \mathcal{I}_{vcc} , $\mathcal{I}_{vcc} = \{\{v_4, v_5, v_9\}, \{v_2, v_8, v_{10}\}, \{v_1, v_3\}\}$. After combining \mathcal{I}_2 with \mathcal{I}_{vcc} , $\mathcal{I}_{vcc} = \{\{v_4, v_5, v_9\}, \{v_2, v_6, v_8, v_{10}\}, \{v_1, v_3, v_7, v_{11}\}\}$. After combining \mathcal{I}_3 with \mathcal{I}_{vcc} , $\mathcal{I}_{vcc} = \{\{v_4, v_5, v_9\}, \{v_2, v_6, v_8, v_{10}, v_{13}\}, \{v_1, v_3, v_7, v_{11}, v_{12}, v_{14}\}\}$, which is returned as result.

Proposition 2: The approximation ratio of Proc. Color is $\log s + 1 + |VCC|(\log \log |VCC|)^2 / \log^3 |VCC|$.

Proof: Let χ_G, χ_{cc} and χ_{vcc} denote the chromatic number of the graph $G, CC(\mathcal{P})$ and $VCC(\mathcal{P})$, respectively. Let α_{cc} and α_{vcc} denote the number of colors to color $CC(\mathcal{P})$ and $VCC(\mathcal{P})$ by Proc. Color, respectively.

For $CC(\mathcal{P})$, we note that the chromatic number of a graph equals to the size of the minimum set of MISs of that can cover the graph. Therefore, for any CC in \mathcal{P} , the heuristic of SetCover can produce a coloring of CC with an approximation ratio $\log s + 1$. Based on Property 3.1, the approximation ratio of coloring all CC 's in \mathcal{P} is also $\log s + 1$. Hence, $\alpha_{cc} \leq (\log s + 1)\chi_{cc}$.

For $VCC(\mathcal{P})$, since we use SampleIS to color the VCC , the approximation ratio of the coloring of VCC is $|VCC|(\log \log |VCC|)^2 / \log^3 |VCC|$. Hence, $\alpha_{vcc} \leq (|VCC|(\log \log |VCC|)^2 / \log^3 |VCC|)\chi_{vcc}$.

Since coloring a graph needs more colors than coloring a subgraph, $\chi_{cc} \leq \chi_G$ and $\chi_{vcc} \leq \chi_G$. It is clear that $(\log s + 1)\chi_{cc} \leq (\log s + 1)\chi_G$ and $(|VCC|(\log \log |VCC|)^2 / \log^3 |VCC|)\chi_{vcc} \leq (|VCC|(\log \log |VCC|)^2 / \log^3 |VCC|)\chi_G$. Hence, $\alpha_{cc} + \alpha_{vcc} \leq (\log s + 1 + |VCC|(\log \log |VCC|)^2 / \log^3 |VCC|)\chi_G$. By Property 3.2, G can be colored with $\alpha_{cc} + \alpha_{vcc}$ colors. ■

We note that the approximation ratio of Proc. Color equals to that of SampleIS in the worst case.

Proposition 3: Given a VP $\mathcal{P} = \{CC_1, \dots, CC_{|G|-|VCC|}, VCC\}$ of a graph G , the time complexity of Proc. Color is $O((|G| - |VCC|)s3^{s/3} + |VCC|^3 + \frac{|G|-|VCC|}{s}\sqrt{2}\Delta_G^{2.5})$.

Proof: Let α_{cc} and α_{vcc} denote the number colors to color $CC(\mathcal{P})$ and $VCC(\mathcal{P})$ by Proc. Color, respectively.

For a CC , it takes $O(3^{\frac{s}{3}})$ time to enumerate all MISs of it. To compute a set of MISs to cover CC , the heuristic of SetCover takes $O(s3^{\frac{s}{3}} \ln 3^{\frac{s}{3}}) = O(s^2 3^{\frac{s}{3}})$ time [17]. Therefore, CC can be colored in $O(s^2 3^{\frac{s}{3}})$ time. It totally takes $O((|G| - |VCC|)s3^{s/3})$ time to color all CC 's.

For the VCC , since the time complexity of SampleIS is cubic, it takes $O(|VCC|^3)$ time to color it.

With reference to the combination step, note that a graph G can be colored by $\Delta_G + 1$ colors by a simple greedy algorithm [14]. Since both our SetCover based coloring method and SampleIS outperform the greedy, $\alpha_{cc} \leq \Delta_G + 1$ and $\alpha_{vcc} \leq \Delta_G + 1$. Therefore, the vertex number of the bigraph B is $O(2\Delta_G)$. Since the Hopcroft-Karp algorithm, which is the best known maximum matching algorithm, takes $O(\sqrt{|V(B)||E(B)|})$ on a graph B , the time complexity to combine the colorings of the VCC and a CC is $O(\sqrt{2}(\Delta_G)^{2.5})$. The time to combine all CC 's is $O(\frac{|G|-|VCC|}{s}\sqrt{2}\Delta_G^{2.5})$.

The total time is hence $O((|G| - |VCC|)s3^{s/3} + |VCC|^3 + \frac{|G|-|VCC|}{s}\sqrt{2}\Delta_G^{2.5})$. ■

Procedure VP_cons

Input: A graph G , component size s
Output: A minimal vertex-cut partition

```

01  $\mathcal{P} = \emptyset, VCC = \emptyset, G' = G$ 
02 while  $|G'| > s$ 
03    $S = \emptyset$ 
04   add a vertex  $v$ , s.t.  $N(v, G')$  is the smallest, to  $S$ 
05   while  $|S| < s$ 
06     pick  $v$  from  $N(S, G')$  s.t.  $N(v, G') \setminus S$  is the smallest
07     add  $v$  to  $S$ 
08    $\mathcal{P} = \mathcal{P} \cup \{S\}$ 
09    $VCC = VCC \cup N(S, G')$ 
10   remove  $S$  and  $N(S, G')$  from  $G'$ 
11    $\mathcal{P} = \mathcal{P} \cup \{G'\}$ 
12 return  $\mathcal{P} \cup \{VCC\}$ 

```

Fig. 5: Procedure VP_cons

C. Vertex-cut partition construction

Proposition 3 presents that when s is fixed, it is desired to minimize the size of the VCC of the vertex-cut partition of G . However, computing an optimum VP is an NP-hard problem.

Theorem 1: Given a graph G and a parameter s , it is NP-hard to construct a VP \mathcal{P} of G such that the size of VCC in \mathcal{P} is minimized.

Proof: (Sketch) Our problem is clearly in NP. We then prove that the minimum balanced α -vertex separator (MBVS) problem, which is NP-hard [18], can be reduced to it. Specifically, given a graph G and a value of α , we set $s = \alpha|V|$. Then, a VP with the minimum VCC is a solution of MBVS. ■

We hence propose a heuristic algorithm to compute a VP of G with a minimal VCC . The main idea is that we use the subgraphs of G that have the minimal neighborhoods as the CC s. Specifically, we use the logic of BFS on G to explore a subgraph S of size s (Lines 03-07). To minimize the neighborhood of $N(S, G)$, at each step of BFS, we pick the vertex adding which incurs the smallest increase of neighbors (Lines 04,06). S is added to \mathcal{P} and $N(S, G')$ is added to VCC (Lines 08-09). S and $N(S, G')$ are removed from G' for the next iteration (Line 10). The algorithm is presented in Fig. 5.

Example 3: Consider the graph G_1 in Fig. 1 and $s = 3$. CC_1 is computed as follows. We first add v_1 to CC_1 because it is one of the vertices of the smallest degree in G_1 and $VCC = \{v_2\}$. Since v_1 only has one neighbor v_2 , we add v_2 to CC_1 and $VCC = \{v_3, v_5\}$. v_2 has two neighbors, v_3 and v_5 . If we add v_2 to CC_1 , $VCC = \{v_4, v_5\}$. If we add v_5 to CC_1 , $VCC = \{v_3, v_6, v_8\}$, which is larger than that of v_2 . Therefore, $CC_1 = \{v_1, v_2, v_3\}$ and $VCC = \{v_4, v_5\}$. The same logic is applied to $G_1 \setminus VCC$. Finally, $CC_2 = \{v_6, v_7, v_{11}\}$ and $CC_3 = \{v_{12}, v_{13}, v_{14}\}$ and $VCC = \{v_4, v_5, v_8, v_9\}$.

IV. COLORING A SET OF GRAPHS

In this section, we study the coloring of a set \mathcal{D} of graphs. Our main idea is to extract the common subgraphs of the graphs in \mathcal{D} and use them as the common CC s of the VPs of the graphs, s.t. the common CC s are only processed once. We propose a vertex-cut partition hierarchy (VPH) to represent the common CC s of the VPs of the graphs. The number of colors used by VPH to color each graph in \mathcal{D} is only slightly larger than that of coloring each graph individually in practice.

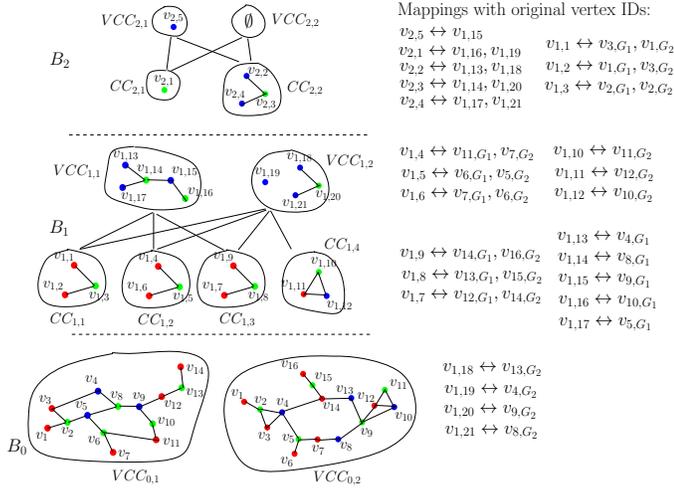


Fig. 6: An example of VP hierarchy

A. Vertex-cut partition hierarchy

We first define a vertex-cut partition bigraph (VPB), which is the building block of our vertex-cut partition hierarchy.

Definition 3: Given a set of graphs \mathcal{D} and a parameter s , let \mathcal{P}_i denote the VP of $G_i \in \mathcal{D}$, the vertex-cut partition bigraph (VPB) is a bigraph $B = (V_{vcc} \cup V_{cc}, E)$, where

- $V_{vcc} = \{VCC_1, VCC_2, \dots, VCC_{|\mathcal{D}|}\}$, $VCC_i = VCC(\mathcal{P}_i)$;
- $V_{cc} = CC(\mathcal{P}_1) \cup CC(\mathcal{P}_2) \cup \dots \cup CC(\mathcal{P}_{|\mathcal{D}|})$;
- $|CC| \leq s$ for any $CC \in V_{cc}$; and
- an edge $(CC, VCC) \in E$ iff there exists a \mathcal{P}_i such that both CC and VCC belong to \mathcal{P}_i .

Let $V_{vcc}(B)$ and $V_{cc}(B)$ denote the sets of nodes in V_{vcc} and V_{cc} of B , respectively. Then, we define the vertex-cut partition hierarchy as follows.

Definition 4: Given a set of graphs \mathcal{D} and a parameter s , the VPH \mathcal{H} is a list of VPBs B_0, B_1, \dots, B_L , where

- $V_{vcc}(B_0) = \{G_1, G_2, \dots, G_{|\mathcal{D}|}\}$, $V_{cc}(B_0) = \emptyset$ and $E(B_0) = \emptyset$;
- B_l is a VPB of the V_{vcc} of B_{l-1} ; and
- $|CC| \leq s$ for any $CC \in \cup_{i=1}^L V_{cc}(B_i)$

Example 4: Fig. 6 presents the VPH of $\mathcal{D} = \{G_1, G_2\}$ with $s = 3$ and $L = 2$. For presentation simplicity, let $VCC_{l,i}$ and $CC_{l,i}$ denote the VCC_i and CC_i of B_l , respectively. $\mathcal{P}_i(B_l)$ denotes the VP of $VCC_{l-1,i}$, which is a subgraph of B_l induced by $VCC_{l,i}$ and its neighboring CC 's in B_l . $\mathcal{P}_i(B_1)$ is the VP of $G_i \in \mathcal{D}$. We reassign the IDs of the vertices for illustration purpose and the mappings with the original IDs are also presented in Fig. 6. B_1 is a VPB of G_1 and G_2 and B_2 is a VPB of $VCC_{1,1}$ and $VCC_{1,2}$. $\mathcal{P}_1(B_1) = \{CC_{1,1}, CC_{1,2}, CC_{1,3}, VCC_{1,1}\}$, $\mathcal{P}_1(B_2) = \{CC_{2,1}, CC_{2,2}, VCC_{2,1}\}$ and $\mathcal{P}_2(B_2) = \{CC_{2,1}, CC_{2,2}, VCC_{2,2}\}$.

Procedure MGColor
Input: VPH \mathcal{H} of a set of graphs \mathcal{D} , vcc_level l
Output: A minimal IS cover for each graph in \mathcal{D}

```

01 for each  $VCC_i$  in  $B_L$ 
02  $\mathcal{I}_i = \text{SampleIS}(VCC_i)$ 
03 while  $l = L$  to 1
04 for each  $CC$  in  $B_l$ 
05  $\mathcal{M}_{CC} = \text{MISEnum}(CC)$ 
06  $\mathcal{I}_{CC} = \text{ISPartition}(\mathcal{M}_{CC})$ 
07 for each  $i = 1$  to  $|\mathcal{D}|$ 
08 let  $\mathcal{P}_i(B_l) = \{CC_1, CC_2, \dots, CC_k, VCC_i\}$ 
09  $\mathcal{I}_i = \text{comb}(\mathcal{I}_i, \mathcal{I}_{CC_1}, \mathcal{I}_{CC_2}, \dots, \mathcal{I}_{CC_k})$ 
10 return  $\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_{|\mathcal{D}|}$ 

```

Fig. 7: Procedure MGColor

B. Coloring algorithm using VPH

Now we present the algorithm to color a set of graphs \mathcal{D} as shown in Fig. 7. Our idea is that for each $i = 1 \dots |\mathcal{D}|$, we iteratively color the $VCC_{i,l}$ using the logic of Proc. MGColor from $l=L$ to 0, as $VCC_{i,0}$ is the G_i in \mathcal{D} . Specifically, Lines 01-02 color each $VCC_{i,L}$ of B_L as the initialization and Lines 03-09 are the main loop. For l from L to 1, Lines 04-06 first color each CC of B_l using our IS partition based method. Then, for each $i = 1 \dots |\mathcal{D}|$, Lines 07-09 combine the colorings of $VCC_{i,l}$ and its neighboring CC 's in B_l to obtain a coloring of $VCC_{i,l-1}$, as $VCC_{i,l}$ and its neighboring CC 's comprise the VP of $VCC_{i,l-1}$. Note that each CC of B_l is just colored once, it can be shared in the VPs of several different VCC 's of B_{l-1} .

Example 5: Consider the VPH shown in Fig. 6. For B_2 , we first color $VCC_{2,1}$ and $VCC_{2,2}$ and obtain $\mathcal{I}_1 = \{\{v_{2,5}\}\}$ and $\mathcal{I}_2 = \{\emptyset\}$. We color the CC 's of B_2 and obtain $\mathcal{I}_{CC_{2,1}} = \{\{v_{2,1}\}\}$ and $\mathcal{I}_{CC_{2,2}} = \{\{v_{2,2}, v_{2,4}\}, \{v_{2,3}\}\}$. After comb (and ID mapping for illustration purpose), the coloring for $VCC_{1,1}$ is $\mathcal{I}_1 = \{\{v_{1,13}, v_{1,15}, v_{1,17}\}, \{v_{1,14}, v_{1,16}\}\}$ and the coloring for $VCC_{1,2}$ is $\mathcal{I}_2 = \{\{v_{1,20}\}, \{v_{1,18}, v_{1,19}, v_{1,21}\}\}$. Note that the color of $v_{2,1}$ changes as $\{v_{2,1}\}$ is merged with $\{v_{2,2}, v_{2,4}\}$. For B_1 , we color the CC 's and obtain $\mathcal{I}_{CC_{1,1}} = \{\{v_{1,1}, v_{1,2}\}, \{v_{1,3}\}\}$, $\mathcal{I}_{CC_{1,2}} = \{\{v_{1,4}, v_{1,6}\}, \{v_{1,5}\}\}$, $\mathcal{I}_{CC_{1,3}} = \{\{v_{1,9}, v_{1,7}\}, \{v_{1,8}\}\}$ and $\mathcal{I}_{CC_{1,4}} = \{\{v_{1,10}\}, \{v_{1,11}\}, \{v_{1,12}\}\}$. Similarly, after comb, we obtain the colorings of G_1 and G_2 . $\mathcal{I}_1 = \{\{v_4, v_5, v_9\}, \{v_1, v_3, v_7, v_{11}, v_{12}, v_{14}\}, \{v_2, v_6, v_8, v_{10}, v_{13}\}\}$ and $\mathcal{I}_2 = \{\{v_1, v_3, v_6, v_7, v_{12}, v_{14}, v_{16}\}, \{v_2, v_5, v_9, v_{11}, v_{15}\}, \{v_4, v_8, v_{10}, v_{13}\}\}$.

Proposition 4: Given a VPH $\mathcal{H} = \{B_0, B_1, \dots, B_L\}$ of a set of graphs \mathcal{D} , the time complexity of Proc. MGColor is $O(s^2 3^{s/3} \sum_{l=1}^L |V_{cc}(B_l)| + \sum_{i=1}^{|\mathcal{D}|} |VCC_{i,L}|^3 + \sqrt{2}(\sum_{i=1}^{|\mathcal{D}|} \Delta_{G_i}^{2,5})(\sum_{l=1}^L |V_{cc}(B_l)|))$.

Proof: As presented in the proof of Theorem 1, it takes $O(s^2 3^{s/3})$ time to color a CC in \mathcal{H} . Hence, it takes $O(s^2 3^{s/3} \sum_{l=1}^L |V_{cc}(B_l)|)$ time to color all CC 's in \mathcal{H} . Since the time complexity of SampleIS is cubic, it takes $O(\sum_{i=1}^{|\mathcal{D}|} |VCC_{i,L}|^3)$ time to color all VCC 's in \mathcal{H} . Regarding the combination step, since the time for combining the colorings of the VCC and a CC of $\mathcal{P}_i(B_l)$ is $O(\sqrt{2} \Delta_{G_i}^{2,5})$ and $\mathcal{P}_i(B_l)$ has at most $|V_{cc}(B_l)|$ CC 's, the combination time for $\mathcal{P}_i(B_l)$ is hence $O(|V_{cc}(B_l)| \sqrt{2} \Delta_{G_i}^{2,5})$. Therefore, the combination time for all $1 \leq i \leq |\mathcal{D}|$ and $1 \leq l \leq L$ is $O(\sqrt{2}(\sum_{i=1}^{|\mathcal{D}|} \Delta_{G_i}^{2,5})(\sum_{l=1}^L |V_{cc}(B_l)|))$. In all, the total time complexity is $O(s^2 3^{s/3} \sum_{l=1}^L |V_{cc}(B_l)| + \sum_{i=1}^{|\mathcal{D}|} |VCC_{i,L}|^3 + \sqrt{2}(\sum_{i=1}^{|\mathcal{D}|} \Delta_{G_i}^{2,5})(\sum_{l=1}^L |V_{cc}(B_l)|))$. ■

```

Procedure VPH_cons
Input: A set of graphs  $\mathcal{D}$ , parameters  $L$  and  $s$ 
Output: VPH  $\mathcal{H}$  of  $\mathcal{D}$ 

01 construct an empty bigraph  $B_0$ 
02 for each  $G_i$  in  $\mathcal{D}$ 
03   insert  $VCC_{i,0} = G_i$  to  $V_{vcc}(B_0)$ 
04 for  $l = 1$  to  $L$ 
05    $B_l = \text{VPB\_cons}(B_{l-1}, s)$ 
06   add  $B_l$  to  $\mathcal{H}$ 
07 return  $\mathcal{H}$ 

function VPB_cons( $B_{l-1}, s$ )
08  $B_l$  is an empty bigraph
09 insert  $VCC_{l,1}, \dots, VCC_{l,|\mathcal{D}|}$  to  $V_{vcc}(B_l)$ ,  $VCC_{l,i}$  is empty
10 for each  $VCC_{l-1,i}$  in  $B_{l-1}$ 
11   insert  $CC = VCC_{l-1,i}$  to  $V_{cc}(B_l)$ 
12   insert edge ( $VCC_{l,i}, CC$ ) to  $E(B_l)$ 
13 let  $\mathcal{A}$  be the set of  $CC$ 's of  $B_l$  whose size exceeds  $s$ 
14 if  $\mathcal{A} \neq \emptyset$ 
15    $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_m = \text{cluster}(V_{cc}(B_l))$ 
16   for each cluster  $\mathcal{C}$ 
17     if  $|\mathcal{C}| > 1$ 
18       compute the maximal weighted common subgraph  $mcs$ 
19       insert  $CC_{mcs} = mcs$  to  $V_{cc}(B_l)$ 
20       for each  $CC$  in  $\mathcal{C}$ 
21         for each neighbor  $VCC$  of  $CC$ 
22           insert edge ( $CC_{mcs}, VCC$ ) to  $E(B_l)$ 
23           add  $N(mcs, CC)$  to  $VCC$ 
24           remove  $mcs$  and  $N(mcs, CC)$  from  $CC$ 
25       else //let  $\mathcal{C} = \{CC\}$ 
26          $\mathcal{P}_{CC} = \text{VP\_cons}(CC, s)$ 
27         for each  $CC' \in CC(\mathcal{P}_{CC})$ 
28           add  $CC'$  to  $V_{cc}(B_l)$ 
29         for each neighbor  $VCC$  of  $CC$ 
30           add  $VCC(\mathcal{P}_{CC})$  to  $VCC$ 
31         for each  $CC' \in CC(\mathcal{P}_{CC})$ 
32           add edge ( $CC', VCC$ ) to  $E(B_l)$ 
33         remove  $CC$  from  $B_l$ 
34       goto 13
35 return  $B_l$ 

```

Fig. 8: **Procedure** VPB_cons

C. VPH construction

Proposition 4 presents that when s is fixed, the governing factor in the time complexity is the number of CC 's and the size of the VCC 's in \mathcal{H} . Therefore, it is desired to minimize the number of CC 's and minimize the size of each VCC 's in \mathcal{H} . However, it is again an NP-hard problem.

Theorem 2: Given a set of graphs \mathcal{D} and the parameters s and L , it is NP-hard to construct a VPH of \mathcal{D} s.t. $\sum_{l=1}^L |V_{cc}(B_l)|$ is minimized and each VCC of B_L is minimized.

Proof: (Outline) We establish that an instance of the problem is equivalent to an NP-hard problem: Consider $L = 1$ and $\mathcal{D} = \{G_1, G_2\}$, where G_1 comprises two connected components C_1 and C_2 and G_2 consists of two connected components C_2 and C_3 . Assume further $|C_1| = |C_2| = |C_3| = s$. Then, constructing the optimum VPH is equivalent to compute the maximum common subgraph between G_1 and G_2 , which is an NP-hard problem. Therefore, constructing the optimum VPH is NP-hard. ■

We propose a heuristic algorithm to construct an optimal VPH, whose rationales can be described as follows. (i) To reduce the number of CC 's, we use as many common CC 's as possible. (ii) To reduce the size of each VCC of B_L , we minimize the number of neighbors of each CC .

The construction algorithm is presented in Fig. 8. Specif-

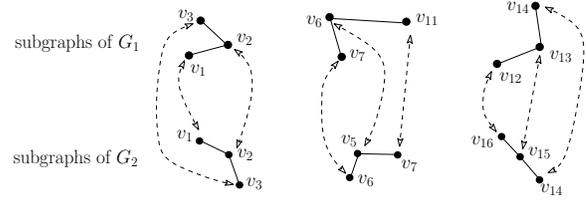


Fig. 9: Mapping of the MCS between G_1 and G_2

ically, Lines 01-03 first initialize the VPB B_0 of \mathcal{D} . Then, Lines 04-06 iteratively construct the VPB B_l of the VCC 's of B_{l-1} using the function VPB_cons. In function VPB_cons, Lines 08-12 initialize B_l , where the VCC 's of B_{l-1} are the CC 's of B_l and the VCC 's of B_l are $|\mathcal{D}|$ empty graphs. Then, Lines 13-34 iteratively partition the CC 's whose size exceeds s , until each CC of B_l is not larger than s , and add the vertex-cut vertices of the partitions to the VCC 's of B_l . When we partition the CC 's, we extract the common subgraphs of the CC 's as the common CC 's of B_l . If we directly compute the maximal common subgraph (MCS) of all CC 's, the size of the MCS may be small. Therefore, Line 15 clusters the CC 's, such that the MCS of the CC 's, denoted as mcs , in each cluster \mathcal{C} is larger than s . The cluster step is a simple greedy algorithm. Given a list of graphs \mathcal{L} and a cluster \mathcal{C} , we iteratively pick the largest graph G in \mathcal{L} , and add G to \mathcal{C} if the MCS of all the graphs in $\mathcal{C} \cup \{G\}$ is larger than s . Output \mathcal{C} until all graphs in the list have been tested and do the same logic on $\mathcal{L} \setminus \mathcal{C}$. Note that mcs is the weighted MCS, where the weight is defined as $|mcs| + \text{avg}_{CC \in \mathcal{C}}(|N(mcs, CC)|)$. mcs is added to $V_{cc}(B_l)$ (Line 19). For each CC in the cluster, we insert an edge from mcs to each neighbor VCC of CC (Line 22). $N(mcs, CC)$ is added to VCC (Line 23), and mcs and $N(mcs, CC)$ are removed from CC (Line 24). The MCS may not be connected. If a cluster just has one CC , we can partition CC by constructing a VP \mathcal{P}_{CC} of CC (Line 26). $VCC(\mathcal{P}_{CC})$ is added to each neighbor of CC in B_l (Line 30). CC is replaced by $CC(\mathcal{P}_{CC})$ (Lines 27-28,33) and we insert an edge from each CC' in $CC(\mathcal{P}_{CC})$ to each neighbor of CC in B_l (Lines 31-32).

Example 6: Let us construct a VPH for $\mathcal{D} = \{G_1, G_2\}$ with $s = 3$ and $L = 2$. B_1 is initialized with $CC_{1,1} = VCC_{0,1} = G_1$, $CC_{1,2} = VCC_{0,2} = G_2$ and two empty VCC 's. We first compute the MCS between G_1 and G_2 , as shown in Fig. 9. We add the MCS to $V_{cc}(B_1)$ and add its neighbors in G_1 and G_2 to $VCC_{1,1}$ and $VCC_{1,2}$, respectively. $CC_{1,2}$ becomes $\{v_9, v_{10}, v_{11}, v_{12}\}$. Since the CC 's in B_1 are still larger than s , they are further partitioned and the result of B_1 is shown in Fig. 6. The same logic is applied to the $VCC_{1,1}$ and $VCC_{1,2}$ to construct B_2 and Fig. 6 shows the result.

D. Determine the optimal values of s and L

Since our VColor involves two parameters s and L , we propose techniques to determine the values of them s.t. the coloring time is as small as possible. Our main idea is that (i) designing a cost model to estimate the coloring time with a VPH; (ii) estimating the coefficients of the cost model by a sampling technique; and (iii) searching for the optimal values of s and L whose estimated coloring time is minimal. The cost model for determining the parameters are presented below.

Definition 5: Given a VPH $\mathcal{H} = \{B_1, \dots, B_L\}$ of \mathcal{D} , the time

TABLE II: Some statistics of datasets

	$ V(G) $	$ E(G) $
Pokec	1.63M	22.30M
PA	1.09M	1.54M
NY	264K	733K
Epinions	75K	508K
Yeast	3.1K	12.5K
LatinSquare	0.9K	307.4K

cost using Proc. MGColor to color the graphs in \mathcal{D} is

$$t = t_{cc} \times \sum_{l=1}^{L-1} |V_{cc}(B_l)| + \sum_{i=1}^{|\mathcal{D}|} t_{vcc,i} + \sum_{l=1}^L \sum_{i=1}^{|\mathcal{D}|} t_{comb,l,i},$$

where t_{cc} is the time to color a CC using our IS partition method, $t_{vcc,i}$ denotes the time to color the $VCC_{L,i}$ of $\mathcal{P}_i(B_L)$ using SampleIS, $t_{comb,l,i}$ denote the time to combine the colorings of the $VCC_{l,i}$ and its neighboring CC 's.

We can estimate the parameters t_{cc} , $t_{vcc,i}$ and $t_{comb,l,i}$ of the cost model by a sampling method. Specifically, we first sample m subgraphs of the size s from the graphs in \mathcal{D} . Then, we color them using our IS partition method and SampleIS, respectively. Let t_1 be the average time of our IS partition method and t_2 be the average time of SampleIS on the m samples. We can set $t_{cc} = t_1$, as the size of each CC is s . We can set $t_{vcc,i} = t_2 \left(\frac{|VCC_{L,i}|}{s}\right)^3$, as the time complexity of SampleIS is cubic and t_2 is the time on a graph of size s . To estimate $t_{comb,l,i}$, the key is to estimate the computation time of the maximum matching of the colorings of VCC of $\mathcal{P}_i(B_l)$ and each CC of $\mathcal{P}_i(B_l)$. We recall that the currently best algorithm to compute the maximum matching of a bigraph is the Hopcroft-Karp algorithm. Therefore, to estimate the combination time of the coloring of the VCC and CC of $\mathcal{P}_i(B_l)$, we can first construct a complete bigraph B_{CC} , where one part has $\Delta_{VCC}+1$ vertices and the other part has $\Delta_{CC}+1$ vertices. We can use the running time of the Hopcroft-Karp algorithm on B_{CC} as the estimation of combining the colorings of the VCC and CC of $\mathcal{P}_i(B_l)$, and sum up the estimated time of each CC in $\mathcal{P}_i(B_l)$ as the estimation of $t_{comb,l,i}$. To search for the optimal values of L and s , we can construct a VPH for each possible values of s and L and estimate the coloring time. Note that the number of possible values of s and L is small in practice.

V. EXPERIMENTAL EVALUATION

In this section, we present an experimental evaluation of our techniques proposed in this paper.

Experimental settings. All the experiments are conducted on a server with an Intel Xeon 2.67GHz CPU and 32GB RAM, running CentOS 5.6. We implement the algorithm in Java 1.7. The popular graph library jgraphT [19] is used in our implementation.

Benchmark datasets. We use six datasets in our experiments: two graphs of small sizes LS and Yeast, two graphs of modest sizes Epinions and NY, and two graphs of large sizes PA and Pokec. LS is a latin square graph, which is often used in graph coloring works. Yeast is a biological network of Yeast [20]. Epinions and Pokec are two social networks. NY and PA are two road networks. They are available at [21]. As the original graphs of Pokec and Epinions are directed, we convert them

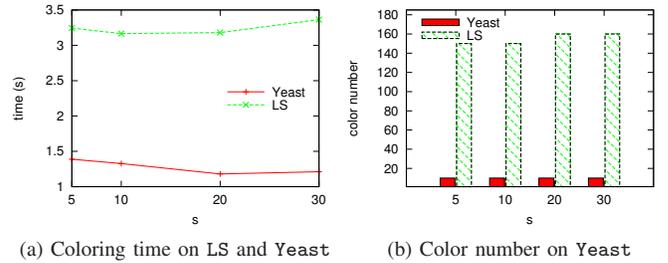


Fig. 10: Results on Yeast

into undirected graphs, by removing the directions of all edges. Some statistics of the graphs are reported in Table II.

The running time of VColor is the total time of the VP or VPH construction times and the coloring time. We use VColor to refer to the whole technique. If we do not recursively partition the VCC , VColor refers to Proc. Color; otherwise, Proc. MGColor. On a set of graphs, VColor always refers to Proc. MGColor. In addition, MGColor with input $\{G\}$ and $L=1$ is equivalent to Color with the input G .

A. Comparison with existing methods

In this subsection, we not only compare VColor with SampleIS, but also compare it with a greedy method Greedy, as it is currently the fastest graph coloring method. Greedy first sorts the vertices of a graph by their degrees and then assigns to a vertex the smallest available color not used by its neighbors ahead of it. The approximation ratio of Greedy is $\Delta+1$, which is worse than SampleIS, but it is much faster than SampleIS, as reported by a recent survey [22]. We compare VColor with SampleIS only on Yeast and LS, as SampleIS cannot finish on other graphs. L of VColor is set to be 1 on Yeast and 3 on Epinions and NY, s.t. the VCCs are small enough to be colored by SampleIS.

SampleIS takes 601 and 2,381 seconds, and Greedy takes 0.2 and 1.9 seconds to color Yeast and LS, respectively. In comparison, VColor can color Yeast in 1.2 seconds and LS in 3.2 seconds when $s = 20$ as shown in Fig 10(a), which is about 500 and 744 times faster than SampleIS, respectively. From Fig 10(a), we also note that the time of VColor is not monotonous with the growth of s . The main reason is that although the number of CCs decreases with the growth of s but the time to color a CC increases.

For the number of colors, SampleIS uses 13 and 137 colors, and Greedy uses 10 and 219 colors on Yeast and LS, respectively. Fig. 10(b) shows that our VColor can color Yeast using 10 colors and LS using 150 colors.

On the graphs of modest sizes, Greedy takes 1.4 seconds to color NY and 0.9 second to color Epinions. As shown in Fig. 11(a)-(b), VColor can color NY in 103 seconds and color Epinions in 8.1 seconds.

For the number of colors, Greedy uses 6 colors on NY and 31 colors on Epinions. As shown in Fig. 11(c), VColor uses fewer than 6 colors on NY and the number of colors is stable with different s . But, the results on Epinions exhibits more variations. As shown in Fig. 11(d), VColor takes 33 colors on Epinions when $s = 10$, which is close to Greedy, but the number of colors increases with the growth of s . The reason

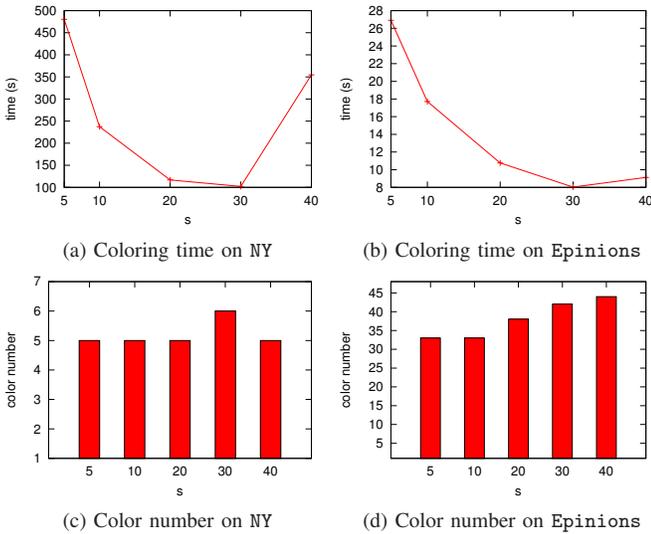


Fig. 11: Results on NY and Epinions

is as follows. Given a graph G , the larger subgraphs of G , including both the vertices and edges, covered by the CCs, VColor can get closer to the global optimal. However, the degrees of vertices in a social network vary significantly. Hence, the number of crossing edges between CCs and the VCC increases with the growth of s . For example, on Epinions, when $s = 10$, the number of crossing edges between CCs and the VCC is 4,650K, whereas it becomes 4,845K when $s = 40$. Therefore, the size of the subgraph covered by the CCs reduces, and more colors are used. However, the marginal increment of the number of colors reduces with the growth of s and the number of colors is no more than 44, as shown in Fig. 11(d).

B. Coloring a large graph

In this subsection, we focus on the performance of our techniques to color a large graph. We use Pokec and PA in this experiment. Greedy on Pokec and PA finishes by 39.1 and 3.7 seconds, and the color number is 48 and 6, respectively. We tune both s and L of VColor in this experiment.

1) *Experiments on the size of VCC*: Recall that given a VPH $\mathcal{H} = \{B_1, B_2, \dots, B_L\}$, our VColor colors the VCC of B_L using SampleIS, which cannot finish in days if the VCC is large. Therefore, we first find a value of L , such that the VCC of B_L is small enough. The results are shown in Fig. 12.

Fig. 12(a)-(b) show that the size of VCC of B_L reduces fastly as the growth of L . Take the results of $s = 20$ as an example. On Pokec, the size of VCC is 187,713 when $L = 5$, it reduces to 1,851 when $L = 15$, and it becomes 0 when $L = 20$. On PA, the size of VCC is 276,148 when $L = 1$, it reduces to 4,804 when $L = 2$, and it becomes 3 when $L = 3$. We note that the size of the VCC of B_L on a social network can be larger than that on a road network for the same value of L . It is because that the average degree of a social network is much larger than that of a road network. From Fig. 12(a)-(b), we also observe that the size of VCC of B_L reduces with the growth of s . It is because that larger CCs can cover more vertices of a graph and the VCC is hence smaller.

2) *Experiments on the coloring time*: In this experiment, we examine $L = 15$ and 20 on Pokec and $L = 3$ and 4 on

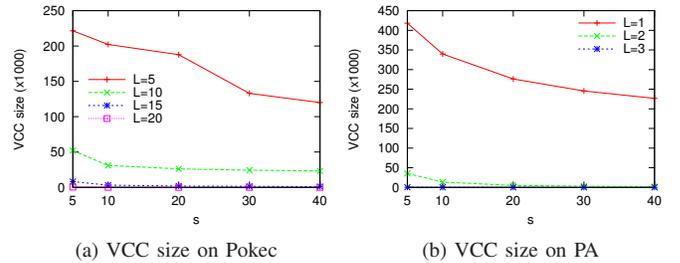


Fig. 12: VCC sizes on large graphs

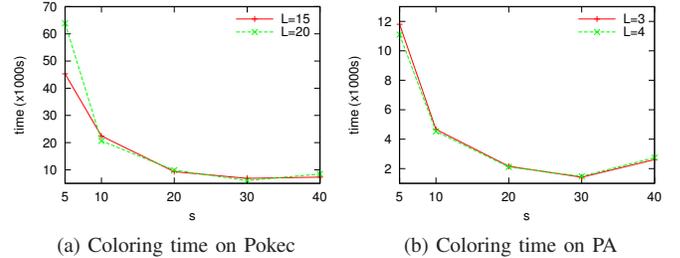


Fig. 13: Time of coloring a large graph

PA, as the sizes of the VCC is small enough to be colored by SampleIS as discussed in Sec. V-B1. The results are shown in Fig. 13.

From Fig. 13(a)-(b), we observe that the coloring time first reduces and then increases with the growth of s . The reason has been explained in Sec. V-A. Fig. 13(a) also shows that the coloring times for $L = 15$ and 20 are very close on Pokec. The reason is that the VPH for $L = 15$ is very close to that for $L = 20$. For example, when $s = 20$, the VPBs $B_{16}, B_{17}, \dots, B_{20}$ totally contain only 1,851 vertices, which is very small compared with the size of Pokec. We have a similar observation on PA as shown in Fig. 13(b).

3) *Experiments on the number of colors*: Following the experiment in Sec. V-B2, we also set $L = 15$ and 20 on Pokec and $L = 3$ and 4 on PA. The results are shown in Fig. 14.

From Fig. 14(a), we observe that the number of colors increases with the growth of s , but the marginal increase reduces on the social network, as explained in Sec. V-A (Fig. 11(d)). From Fig. 14(a), we also observe that the number of colors slightly increases with the growth of L . The reason is that the VCC of B_{15} is further partitioned for $L = 20$, and hence the VCC of B_{15} can be colored using slightly more colors by VColor than directly applying SampleIS on it. From Fig. 14(b), we observe that the number of colors is very stable with the growth of s and L on the road network PA.

C. Coloring a set of graphs

This subsection presents the experimental results on coloring a set of graphs. We examine the performance of MGColor on three datasets: PubChem, PA and Pokec. For PubChem, since many graphs of it are very small (e.g., the smallest graph just has 2 vertices) and our vertex-cut partition based method cannot show its advantage on very small graphs, we randomly pick 8k graphs whose size is larger than 50 from PubChem as the tested dataset. The smallest and largest graph tested have 50 and 419 vertices, respectively, and the std. dev. of the graph sizes is 15.1. For PA and Pokec, to obtain a large number of

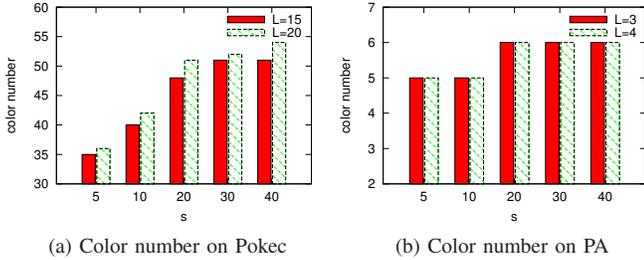


Fig. 14: Performance of coloring a large graph

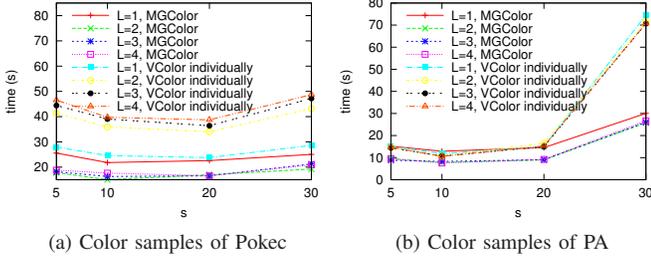


Fig. 15: Time of coloring 800 samples of Pokec and PA

graphs, we randomly sample a set of 800 connected subgraphs of size 1k from PA and Pokec, respectively.

1) *Experiments on the coloring time:* For PubChem, since the graphs are small, we set $L = 1$ and $s = 5$. The runtime of MGCOLOR on the tested dataset is 5.3 seconds. The results on the samples of PA and Pokec are shown in Fig. 15. From Fig. 15(a), we observe that the time to color the Pokec samples first reduces and then increases with the growth of s . The trend is consistent to the coloring time on a single graph. From Fig. 15(a), we also observe that the coloring time is different for different values of L . Similar results can be observed on PA as shown in Fig. 15(b).

To show the advantage of MGCOLOR on coloring a set of graphs, we also show the runtime of coloring each graph one by one using Proc. Color. On PubChem, Color totally takes 7.2 seconds, which is 1.4 times slower than MGCOLOR. Color is 2 and 2.5 times slower than MGCOLOR on Pokec and PA, respectively, as shown in Fig. 15(a)-(b).

2) *Experiments on the number of colors:* Since we color a set of graphs, we cannot directly show the number of colors of all graphs. We hence show the average distance of the number of colors of MGCOLOR with that of coloring each graph individually using VColor. The distance is defined as $dist = avg_{G \in \mathcal{D}} (a/b - 1)$, where a is the color no. of G computed by MGCOLOR, b is the color no. of G computed by VColor and \mathcal{D} is a set of graphs.

The results show that MGCOLOR may use slightly more colors than coloring each graph individually. The reason is that given a set of graphs \mathcal{D} , VPH needs to maximize the common CCs when constructing VPs for the graphs in \mathcal{D} . Therefore, for a graph G in \mathcal{D} , the VP of G in VPH may not as efficient as the VP for G itself. However, the overhead of color number is small. For example, on PubChem with $L = 1$ and $s = 5$, the color distance is 0.09. On Pokec with $L = 2$ and $s = 20$, the average number of colors of the 800 graphs is 11.3 and the color distance is 0.05, as shown in Fig. 16. The overhead of color number is hence no more than 1.

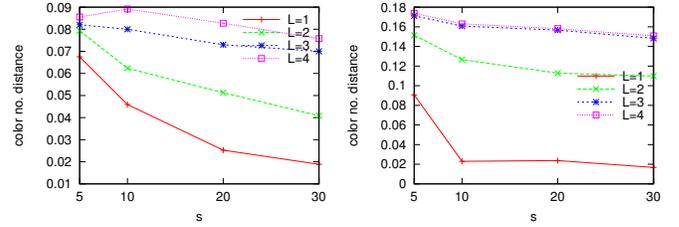


Fig. 16: Color number distance from MGCOLOR to coloring each sample individually

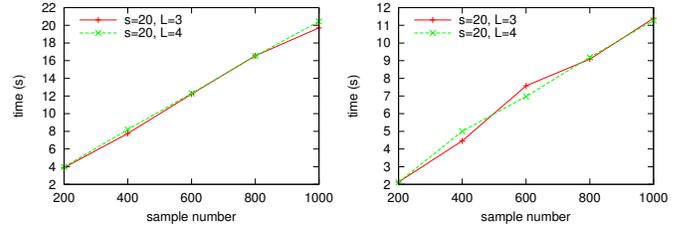


Fig. 17: Scalability performance of MGCOLOR

Fig. 16 shows that the distance reduces with the growth of s . The reason is that our vertex-cut partition logic prefers to include the vertices of higher degree into the VCC when s is large, as discussed in Sec. V-A. However, a graph usually has few vertices of high degree. Therefore, the VP of G in VPH gets closer to the VP constructed for G itself, when s is large.

Fig. 16 also shows that the distance increases with the growth of L . The reason is that each level of VPH introduces some distance of color number. If we increase the number of levels, the distance aggregates. However, the marginal increase of distance reduces with the growth of L . For instance, the distances of $L = 3$ and $L = 4$ are almost the same.

3) *Experiments on scalability:* In this experiment, we examine the scalability of MGCOLOR by coloring 400, 600, 800 and 1,000 sample subgraphs of Pokec and PA, respectively. Fig. 17 presents the results. From Fig. 17, we observe that the time of MGCOLOR grows almost linearly with the increase of the number of graphs.

4) *Experiments on the accuracy of cost model:* In this experiment, we present the accuracy of our cost model for estimating the values of s and L that can produce the smallest coloring time. The results are presented in Fig. 18. We test several combinations of values of s and L , i.e., $s = 5, 10, 20, 30$ and $L = 3, 4$. We examine the coloring time of all combinations of these values, and compare the smallest coloring time with the coloring time using the best values estimated by the cost model. Fig. 18 shows that on Pokec and PA, using the best values of s and L estimated by our cost model, the coloring time is only at most 15% and 18% longer than the smallest, respectively.

VI. RELATED WORK

In this section, we present the works that are closely related to this paper. We mainly discuss the approximation methods and the readers who are interested in exact solutions please refer to surveys [8], [10].

Most existing approximation algorithms fall to three frameworks. The first framework is local search heuristics. The

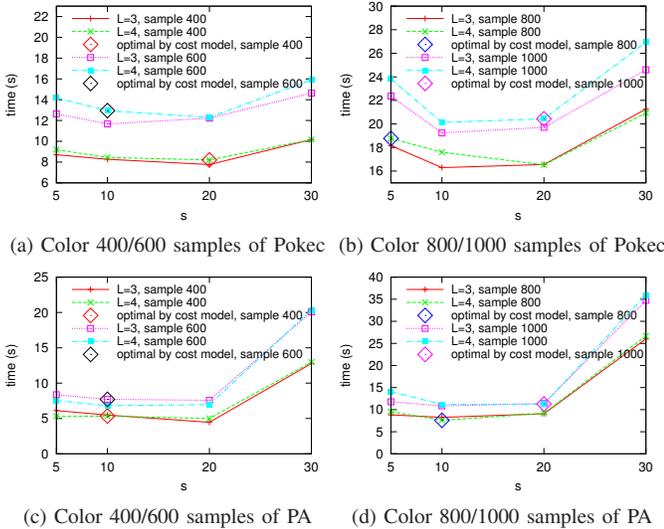


Fig. 18: Performance of cost model

main idea of local search is to iteratively choose a “move” of a coloring (*i.e.*, changing the color of a vertex) that can decrease the cost function, until a local optimum is reached. A tabu algorithm TABUCOL [5] is a seminal work of local search and it is still the foundation of many recent local search methods. For example, considering the static tabu list of TABUCOL may not capture well the neighborhood changes throughout the search, Blchliker and Zufferey [23] design a technique that can support a dynamic tabu list throughout the search. Considering TABUCOL assigns the same cost to all edge violations, which may result in a poor local optimum, Porumbel et al. [24] modify the cost function of TABUCOL by assigning different costs to different edge violations. Hertz et al. [25] design a hybrid method, which integrates TABUCOL and its two variations. The hybrid method outperforms the individual variations on many instances. There are also some simulated annealing based local search algorithms [26]. While the local search heuristics generally support coloring of small and modest size graphs well, their results are often far from the optimum of large graphs [8].

The second framework is evolutionary approach. The main idea is to use colorings as individuals and to recombine the individuals to pass good information to the offsprings. The key of the evolution is the recombination operator. Earlier works use the standard uniform crossover for recombination. For example, Fleurent and Ferland [27] assign to a vertex the color of either the first parent or the second parent. However, the evolutionary approach is not as competitive as local search, until the GPX method [7] is proposed. Instead of using color value of vertices as individuals, GPX proposes to use color classes as individuals and passes the color classes from parents to offsprings in an alternate manner. This idea significantly improved the coloring of this framework. Because of its effectiveness, many recent works follow it. For example, Galinier et al. [28] propose to combine conflict-free color classes from parents. When selecting color classes to pass to offsprings, Porumbel et al. [29] propose to consider both the sizes of color classes and the conflicts. Lu and Hao [30] propose to use several parents in evolution. However, using several parents is only useful on the graphs having large class sizes. Evolutionary methods can produce good coloring results

on large graphs. But, they are often time consuming.

The third framework is independent set extraction. It is the most promising framework for coloring large graphs [8]. SampleIS [6] used in the experiments follows this framework. This framework comprises two phases: a preprocessing phase and a coloring phase. The preprocessing phase is to iteratively extract a large independent set from the input graph until the residual graph is small enough. Each IS extracted is assigned a unique color. The coloring phase uses existing methods (*e.g.*, TABUCOL) to color the residual graph. The color classes of the residual graph and the ISs extracted give a coloring of the input graph. Many methods for computing large ISs have been proposed, such as simple greedy [31], tabu search [27], XRLF heuristic [32] and sampling [6]. To obtain a smaller residual graph, Wu and Hao [33] propose to extract a set of disjoint independent sets in each iteration, instead of extracting one independent set in each iteration. Recently, there is a trend of having a postprocessing phase, which reconsiders the color of a vertex [34], [35]. It is based on the observation that the large ISs extracted in the preprocessing phase may not belong to the final coloring [33]. The main idea is to add back the ISs extracted to the residual graph and re-color the residual graph starting with its current coloring extended with the added ISs as new color classes.

There are some works that do not belong to the three frameworks. Karger et al. [9] model the graph coloring problem with semidefinite programming. Their techniques can color an α -colorable graph with $\min\{\tilde{O}(\Delta^{1-2/\alpha}), \tilde{O}(|G|^{1-3/(\alpha+1)})\}$ colors. Although it is better than SampleIS in terms of $|G|$, the bound is not specified definitely, as \tilde{O} hides lower-order factors, such as $\log|G|$. Karger et al. mainly focus on α -colorable graphs and acknowledge that SampleIS has the best known approximation ratio for general graphs. There are some works studying parallel or distributed coloring of a graph (*e.g.*, [36], [37]). In contrast, we focus on algorithms that run on a single off-the-shelf machine.

Finally, we end this section with a brief discussion of the related work of multi-query optimization (MQO). The idea of MQO is to share the results of common subqueries (of a set of queries) [38]. MQO significantly reduces the overall computation of graph queries [39], [40], [41]. Le et al. study the MQO of SPARQL queries [39]. They use the maximum common subgraphs as the common subqueries. The results of the common subqueries are combined with the results of the remaining subqueries to obtain the final query results. Hong et al. [40] study MQO of XQuery and Bruno et al. [41] study MQO of path queries on XML. However, these works cannot be directly adopted to solve the graph coloring problem.

VII. CONCLUSION

In this paper, we study the graph coloring problem of large graphs. We propose a vertex-cut partition based coloring method VColor. It partitions a large graph G into a set of connected components (CCs) by removing a vertex-cut component (VCC). The CCs are colored by an MIS enumeration based technique and the VCC is colored by SampleIS. The local colorings are combined by a maximum matching based technique to obtain a minimal coloring of G . VColor is significantly faster than SampleIS, while the numbers of colors used in practice are comparable. We also propose techniques to

optimize the coloring of a set of graphs. A VPH is designed to represent the common subgraphs of the graphs as the common CCs. Our technique is significantly faster than coloring graphs individually. To estimate the parameters' values of VColor for minimal coloring time, we propose a cost model for the coloring time. An efficient sampling based search method is proposed to search for the optimal values of the parameters. Our experiments verify the effectiveness and efficiency of the proposed techniques.

ACKNOWLEDGMENT

Thanks to the support of NSF of China 61502258, 71171122, 71402083, NSF of Shandong Province ZR2014FQ007, GRF12201315, FRG2/13-14/073, the Key Projects of Fundamental Research Program of STCSM 14JC1400300, Singapore MoE AcRF Tier 2 grant MOE2012-T2-2-067.

REFERENCES

- [1] A. Ingrid, "Nucleic acid sequence design as a graph colouring problem," *Ph.D. thesis, Universitt Wien*, pp. 1 – 83, 2005.
- [2] T. Park and C. Y. Lee, "Application of the graph coloring algorithm to the frequency assignment problem," *Journal of the Operations Research*, pp. 258 – 265, 1996.
- [3] G. Chaitin, "Register allocation and spilling via graph coloring," *SIGPLAN Not.*, vol. 39, no. 4, pp. 66–74, 2004.
- [4] V. Lotfi and S. Sarin, "A graph coloring algorithm for large scale scheduling problems," *Computers & Operations Research*, vol. 13, no. 1, pp. 27 – 32, 1986.
- [5] A. Hertz and D. de Werra, "Using tabu search techniques for graph coloring," *Computing*, vol. 39, no. 4, pp. 345–351, 1987.
- [6] M. M. Haldrsson, "A still better performance guarantee for approximate graph coloring," *Information Processing Letters*, vol. 45, no. 1, pp. 19 – 23, 1993.
- [7] P. Galinier and J.-K. Hao, "Hybrid evolutionary algorithms for graph coloring," *Journal of Combinatorial Optimization*, vol. 3, no. 4, pp. 379–397, 1999.
- [8] P. Galinier, J.-P. Hamiez, J.-K. Hao, and D. Porumbel, "Recent advances in graph vertex coloring," in *Handbook of Optimization*, 2013, vol. 38, pp. 505–528.
- [9] D. Karger, R. Motwani, and M. Sudan, "Approximate graph coloring by semidefinite programming," *J. ACM*, vol. 45, no. 2, pp. 246–265, 1998.
- [10] P. Pardalos, T. Mavridou, and J. Xue, "The graph coloring problem: A bibliographic survey," in *Handbook of Combinatorial Optimization*, 1999, pp. 1077–1141.
- [11] "PubChem," <https://pubchem.ncbi.nlm.nih.gov/help.html>.
- [12] Y. Peng, Z. Fan, B. Choi, J. Xu, and S. Bhowmick, "Authenticated subgraph similarity search in outsourced graph databases," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 27, no. 7, pp. 1838–1860, 2015.
- [13] Z. Fan, Y. Peng, B. Choi, J. Xu, and S. Bhowmick, "Towards efficient authenticated subgraph query service in outsourced graph databases," *Services Computing, IEEE Transactions on*, vol. 7, no. 4, pp. 696–713, 2014.
- [14] 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, no. 1, pp. 85–86, 1967.
- [15] D. Eppstein, "All maximal independent sets and dynamic dominance for sparse graphs," *ACM Trans. Algorithms*, vol. 5, no. 4, pp. 38:1–38:14, 2009.
- [16] J. M. Byskov, "Enumerating maximal independent sets with applications to graph colouring," *Oper. Res. Lett.*, vol. 32, no. 6, pp. 547–556, 2004.
- [17] M. Cygan, ukasz Kowalik, and M. Wykurz, "Exponential-time approximation of weighted set cover," *Information Processing Letters*, vol. 109, no. 16, pp. 957 – 961, 2009.
- [18] U. Feige and M. Mahdian, "Finding small balanced separators," in *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, ser. STOC, 2006, pp. 375–384.
- [19] "jgrapht," <http://jgrapht.org/>.
- [20] J. Lee, W.-S. Han, R. Kasperovics, and J.-H. Lee, "An in-depth comparison of subgraph isomorphism algorithms in graph databases," *Proc. VLDB Endow.*, vol. 6, no. 2, pp. 133–144, 2012.
- [21] "SNAP," <http://snap.stanford.edu/data/index.html>.
- [22] T. Husfeldt, *Graph colouring algorithms*, ser. Encyclopedia of Mathematics and its Applications, 2015, ch. 13, pp. 277–303.
- [23] I. Blichler and N. Zufferey, "A graph coloring heuristic using partial solutions and a reactive tabu scheme," *Computers & Operations Research*, vol. 35, no. 3, pp. 960 – 975, 2008.
- [24] D. Porumbel, J.-K. Hao, and P. Kuntz, "A study of evaluation functions for the graph k-coloring problem," in *Artificial Evolution*, ser. Lecture Notes in Computer Science, 2008, vol. 4926, pp. 124–135.
- [25] A. Hertz, M. Plumettaz, and N. Zufferey, "Variable space search for graph coloring," *Dis. App. Math.*, vol. 156, no. 13, pp. 2551 – 2560, 2008.
- [26] C. Morgenstern and H. Shapiro, "Coloration neighborhood structures for general graph coloring," ser. SODA, 1990, pp. 226–235.
- [27] C. Fleurent and J. Ferland, "Genetic and hybrid algorithms for graph coloring," *Annals of Operations Research*, vol. 63, no. 3, pp. 437–461, 1996.
- [28] P. Galinier, A. Hertz, and N. Zufferey, "An adaptive memory algorithm for the k-coloring problem," *Discrete Applied Mathematics*, vol. 156, no. 2, pp. 267 – 279, 2008.
- [29] D. C. Porumbel, J.-K. Hao, and P. Kuntz, "An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring," *Computers & Operations Research*, vol. 37, no. 10, pp. 1822 – 1832, 2010.
- [30] Z. L and J.-K. Hao, "A memetic algorithm for graph coloring," *European Journal of Operational Research*, vol. 203, no. 1, pp. 241 – 250, 2010.
- [31] M. Chams, A. Hertz, and D. de Werra, "Some experiments with simulated annealing for coloring graphs," *European Journal of Operational Research*, vol. 32, no. 2, pp. 260 – 266, 1987.
- [32] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partitioning," *Oper. Res.*, vol. 39, no. 3, pp. 378–406, 1991.
- [33] Q. Wu and J.-K. Hao, "Coloring large graphs based on independent set extraction," *Computers & Operations Research*, vol. 39, no. 2, pp. 283 – 290, 2012.
- [34] —, "An extraction and expansion approach for graph coloring," *Asia-Pacific Journal of Operational Research*, vol. 30, no. 05, p. 1350018, 2013.
- [35] J.-K. Hao and Q. Wu, "Improving the extraction and expansion method for large graph coloring," *Discrete Appl. Math.*, vol. 160, no. 16-17, pp. 2397–2407, 2012.
- [36] A. H. Gebremedhin and F. Manne, "Scalable parallel graph coloring algorithms," *Concurrency: Practice and Experience*, vol. 12, no. 12, pp. 1131–1146, 2000.
- [37] N. Gandhi and R. Misra, "Performance comparison of parallel graph coloring algorithms on bsp model using hadoop," ser. ICNC, 2015, pp. 110–116.
- [38] T. K. Sellis, "Multiple-query optimization," *ACM Trans. Database Syst.*, vol. 13, no. 1, pp. 23–52, 1988.
- [39] W. Le, A. Kementsietsidis, S. Duan, and F. Li, "Scalable multi-query optimization for sparql," ser. ICDE, 2012, pp. 666–677.
- [40] M. Hong, A. J. Demers, J. E. Gehrke, C. Koch, M. Riedewald, and W. M. White, "Massively multi-query join processing in publish/subscribe systems," ser. SIGMOD, 2007, pp. 761–772.
- [41] N. Bruno, L. Gravano, N. Koudas, and D. Srivastava, "Navigation- vs. index-based xml multi-query processing," ser. ICDE, 2003, pp. 139–150.