Reduction and Local Search for Weighted Graph Coloring Problem

Yiyuan Wang,[1,4] **Shaowei Cai,**[2,*] **Shiwei Pan,**[1] **Ximing Li,**[3,4] **Minghao Yin**[1,4*]
[1]School of Computer Science and Information Technology, Northeast Normal University, China
[2]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
[3]College of Computer Science and Technology, Jilin University, China
[4]Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, China
yiyuanwangjlu@126.com, caisw@ios.ac.cn, {pansw779, ymh}@nenu.edu.cn, liximing86@gmail.com

Abstract

The weighted graph coloring problem (WGCP) is an important extension of the graph coloring problem (GCP) with wide applications. Compared to GCP, where numerous methods have been developed and even massive graphs with millions of vertices can be solved well, fewer works have been done for WGCP, and no solution is available for solving WGCP for massive graphs. This paper explores techniques for solving WGCP, including a lower bound and a reduction rule based on clique sampling, and a local search algorithm based on two selection rules and a new variant of configuration checking. This results in our algorithm RedLS (Reduction plus Local Search). Experiments are conducted to compare RedLS with the state-of-the-art algorithms on massive graphs as well as conventional benchmarks studied in previous works. RedLS exhibits very good performance and robustness. It significantly outperforms previous algorithms on all benchmarks.

Introduction

Graph coloring problem (GCP) is a well-known combinatorial optimization problem. In classical GCP, a basic assumption is that vertices in the graph are equally important, however, it is hard to hold in many real world scenarios where each vertex is associated with various types of weights. The paradigm of dealing with such vertex weighted graph refers to the weighted graph coloring problem (WGCP) as a form of vertex colouring, which is also known as the weighted vertex coloring problem or max-coloring problem. Formally, WGCP aims to partition all the vertices into several disjoint subsets such that the sum of those subset costs is minimized, where the cost of each subset is given by the maximum weight of a vertex within the current subset. WGCP has been widely used in many fields (Ribeiro, Minoux, and Penna 1989; Hochbaum and Landy 1997; Gavranovic and Finke 2000; Pemmaraju, Raman, and Varadarajan 2004). Also, WGCP can be directly encoded as the maximum weight stable set problem (MWSS) (Cornaz, Furini, and Malaguti 2017).

WGCP is an NP-hard problem (Garey and Johnson 1979). Assuming that NP≠ZPP, the best approximation ratio of WGCP is $n^{1-\varepsilon}$ for all $\varepsilon > 0$ (Zuckerman 2006), resulting in intractable computations. Although GCP is extensively studied and numerous methods (Verma, Buchanan, and Butenko 2015; Peng et al. 2016; Lin et al. 2017; Zhou, Duval, and Hao 2018; Hébrard and Katsirelos 2018; 2019) have been developed to solve GCP on graphs with various types and sizes, the GCP methods cannot be directly applicable for WGCP due to its additional features and hardness. Naturally, WGCP nowadays is still a challenging problem.

During the past decades, many efforts have been made for handling WGCP. Several exact algorithms have been designed to solve WGCP. For example, Ribeiro et al. (1989) combined branch and bound as well as column generation techniques in a very efficient manner to solve WGCP. The branch and price based WGCP algorithm (Furini and Malaguti 2012) followed a classical rule from traditional GCP to deeply exploit the problem structure. Besides, a recent work (Hsu and Chang 2016) provided the upper bound on the smallest number of colors needed in an optimal WGCP solution in terms of the ratio of the maximum vertex weight and the minimum vertex weight. Those exact WGCP algorithms can prove the optimality of the obtained solutions, but they may fail to solve the problem within reasonable time, especially for massive graphs.

To address this, some approximate but fast heuristic WGCP algorithms have been developed. A greedy randomized adaptive search procedure GRASP was designed for finding approximate solutions to WGCP, which used constructive and destructive moves and a filtering technique (Prais and Ribeiro 2000). A two-phase algorithm (Malaguti, Monaci, and Toth 2009), called 2Phase, generated a large number of feasible solutions by fast greedy heuristics in the first phase and computed the solutions using a Lagrangian-based heuristic method in the second phase. According to the literature, the current best heuristic algorithm for WGCP, namely AFISA (Sun et al. 2018), relied on a mixed search strategy exploring both feasible and infeasible solutions. Although the AFISA has empirically achieved competitive performance on conventional graphs, it involves high-complexity heuristics, making it less efficient for massive graphs such as social networks (Rossi and

*Corresponding author

Ahmed 2015).

Motivated to contribute to solving WGCP problems with massive graphs, in this paper we propose a novel efficient WGCP algorithm, namely RedLS (Reduction plus Local Search). RedLS consists of two stages as below. In the first stage, we propose a lower bound and a novel reduction rule, which are based on clique sampling that will be introduced later. The iterated application of the reduction rule results in a reduction procedure, which is empirically shown to be able to significantly reduce the size of the graphs. In the second stage, we design a novel local search algorithm. There are two main ideas in our local search algorithm. Firstly, we define some candidate operation sets and propose two selection rules to decide which operation should be selected effectively. Secondly, we introduce a new variant of configuration checking (CC) (Cai, Su, and Sattar 2011; Wang, Cai, and Yin 2016; Wang et al. 2018) to deal with the serious cycling problem of local search. During the local search procedure, this strategy forbids the algorithm moving some candidate vertices.

We conduct a number of experiments to evaluate RedLS on conventional benchmarks used in previous studies as well as a benchmark of massive graphs. Experimental results indicate that RedLS almost always finds better solutions than other previous state-of-the-art WGCP algorithms.

In the next section, we introduce some necessary background knowledge. Then, we introduce a lower bound and a novel reduction rule in Section 3. Section 4 describes the local search algorithm and relevant proposed ideas. Experimental evaluations of the RedLS algorithm are shown in Section 5. Finally, we give some concluding remarks.

Preliminaries

Definitions and notations

Let $G = (V, E)$ be an undirected graph where V=$\{v_1, v_2, \ldots, v_n\}$ is the set of vertices and $E = \{e_1, e_2, \ldots, e_m\}$ is the set of edges. Each edge is a 2-element subset of V. For an edge $e = (v, u)$, we say vertices v and u are the *endpoints* of edge e. A vertex weighted graph $G = (V, E, w)$ is an undirected graph where each vertex $v \in V$ is associated with a positive weight $w(v)$. The neighborhood of vertex v is $N(v) = \{u \in V | (v, u) \in E\}$, and the degree of vertex v is defined as $d(v) = |N(v)|$.

A feasible coloring S is a partition of the vertex set V into independent sets $S = \{V_1, V_2, \ldots, V_k\}$ such that no two adjacent vertices are in the same V_i (i.e., $1 \leq i \leq k$). Thus, $|S|$ is the number of colors. The weighted graph coloring problem (WGCP) is to find a feasible coloring $S = \{V_1, V_2, \ldots, V_k\}$ which minimizes $cost(S) = \sum_{i=1}^{k} max_{v \in V_i} w(v)$. Notice that the number of colors is unknown before the optimal solution is found.

During the local search algorithm, we maintain a partition of the vertex set V. Generally, any partition $S = \{V_1, V_2, \ldots, V_k\}$ of the vertex set V is a candidate solution of WGCP. For any candidate solution S, an edge is a conflict edge if its two endpoints appear in the same V_j ($1 \leq j \leq k$), and $CE(S) = \{e'_1, \ldots, e'_t\}$ is the set of conflict edges under S. A candidate solution S is feasible iff $CE(S) = \emptyset$.

For a vertex weighted graph G, a weighted clique C of G is a subset of V where each pair of vertices in C is adjacent, and a weighted clique is maximal if it is not included in a clique with a bigger weight.

Conflict Value and Scoring Function

For a candidate solution S, we define its conflict value as the number of conflict edges. When the algorithm is equipped with an edge weighting mechanism, this concept is generalized to take into account the edge weights. Edge weighting belongs to constraint weighting techniques, which are usually used to diversify search. Our algorithm also uses an edge weighting mechanism, which associates an additional property (i.e., edge weight) $w_e(e_i)$ to each edge e_i. The edge weights are all initialized as 1 and updated during the search. Considering edge weights, the conflict value of a candidate solution S is defined as

$$g(S) = \sum_{e'_i \in CE(S)} w_e(e'_i).$$

Obviously, S is a feasible coloring iff $g(S) = 0$. Suppose S is a candidate solution, operation $\langle v, V_i, V_j \rangle$ is defined as moving vertex v from its color class V_i to a different color class V_j, which leads to a neighboring candidate solution of S. We use $score(v, V_i, V_j)$ to denote the change on the conflict value of $g(S)$ for operation $\langle v, V_i, V_j \rangle$. Formally,

$$score(v, V_i, V_j) = g(S) - g(S'),$$

where $S' = S \oplus \langle v, V_i, V_j \rangle$ is obtained from S by moving vertex v from color class V_i to color class V_j. Therefore, if the algorithm performs an operation with a positive score, the conflict value would be decreased.

Lower Bound and Reduction Rule

In this section, we introduce a lower bound method and a novel reduction rule for WGCP. Both the methods rely on a technique of sampling maximal weighted cliques.

Clique Sampling and Formal Notions

Both the lower bound and the reduction rule are based on clique sampling. We first introduce the algorithm used for sampling maximal weighted cliques, and the key concepts that will be used in the lower bound and reduction rule.

We apply a simplified version of FastWCIq (Algorithm 1) (Cai and Lin 2016) to get some maximal weighted cliques. The simplified FastWCIq samples some random vertices from V (line 1). In our work, $|StartSet| = |V|/100$. During each iteration (lines 2-8), FastWCIq obtains a maximal weighted clique C and puts it into $CliSet$ (line 8). In a weighted clique construction procedure, the algorithm first picks a random vertex from $StartSet$ as the starting vertex from which a weighted clique is extended (lines 5-7). At last, the algorithm returns all found maximal weighted cliques $CliSet = \{C_1, C_2, \ldots, C_t\}$ where $t = |StartSet|$. [1]

[1] We allow two cliques in $CliSet$ share the same vertices and even some cliques in $CliSet$ are the same maximal clique.

Algorithm 1: a simplified version of FastWClq (G)

Input: a weighted graph $G = (V, E, w)$
Output: maximal weighted cliques $CliSet$

$CliSet := \emptyset$, $StartSet :=$ some random vertices from V;
while $StartSet \neq \emptyset$ **do**
  pick a random vertex v from $StartSet$;
  $StartSet := StartSet \setminus \{v\}$, $C := \{v\}$, $CandSet := N(v)$;
  while $Candset \neq \emptyset$ **do**
    select u with the biggest $\sum_{u' \in (N(u) \cap CandSet)} w(u')$ value and $C := C \cup \{u\}$;
    $CandSet := (CandSet \setminus \{u\}) \cap N(u)$;
  $CliSet := CliSet \cup C$;
return $CliSet$;

In the following, we propose an operation $\otimes$ on weighted cliques, which is important to the lower bound and reduction rule. To give the definition, it is necessary to first arrange the cliques in a proper way. Let l denote the maximum clique size (number of vertices) in $CliSet$. For any clique in $CliSet$ with size smaller than l, we put virtual vertices with weight 0 into the clique to make its size become l. Also, the positions of vertices in a clique are arranged in a descending order of the weight values. Formally, the operation $\otimes$ is defined as follows:

Definition 1 *For some weighted cliques $C_1, C_2, \ldots, C_t$, $S_M = C_1 \otimes C_2 \otimes \ldots \otimes C_t = \{V_1, V_2, \ldots, V_l\} = \{\{v_1^1, v_1^2, \ldots, v_1^t\}, \ldots, \{v_l^1, v_l^2, \ldots, v_l^t\}\}$, s.t.*

(i) $V' = C_1 \cup C_2 \ldots \cup C_t$;
(ii) $l = max_{1 \leq i \leq t}\{|C_i|\}$;
(iii) $C_i = \{v_1^i, v_2^i, \ldots, v_{|C_i|}^i\}$;
(iv) $\forall i \in [1, t]$, $w(v_1^i) \geq w(v_2^i) \geq \ldots \geq w(v_{|C_i|}^i)$, *and* $w(v_{|C_i|+1}^i)$=$w(v_{|C_i|+2}^i)$=…=$w(v_l^i)$=0 *if* $|C_i| < l$.

Note that we allow that vertex $v \in V'$ can appear in different subsets $V_i, V_j \in S_M$, for $1 \leq i, j \leq l$. Thus, $S_M = \{V_1, V_2, \ldots, V_l\}$ is called a *"relaxed" partition set* of V' and $cost(S_M) = \sum_{i=1}^{l} max_{v \in V_i} w(v)$.

Lower Bound

This subsection introduces a lower bound for WGCP.

Proposition 1 *Given a vertex weighted graph G=(V, E, w), maximal weighted cliques $CliSet$=$\{C_1, C_2, \ldots, C_t\}$ in it, and $S_M = C_1 \otimes C_2 \ldots \otimes C_t$, then $cost(S_M)$ is a lower bound of WGCP on G (i.e., the cost of optimal solution is at least $cost(S_M)$).*

Proof: Suppose that there exists a better solution S' with $cost(S') < cost(S_M)$. The first difference between S' and S_M is the h^{th} vertex in C_r ($h \geq 1$, $r \geq 1$). Thus, we denote $S' = \{V_1', V_2', \ldots, V_z'\} = \{.., \{v_h^1, \ldots, v_h^{r-1}, v_b^r, ..\}, \ldots, \{v_x^1, \ldots, v_x^{r-1}, v_h^r, ..\}, \ldots\}$. Then, we define $S'' = \{V_1'', V_2'', \ldots, V_z''\}$, which is almost the same as S', with the following two exceptions: $V_h'' = V_h' \setminus \{v_b^r\} \cup \{v_h^r\}$ and $V_x'' = V_x' \setminus \{v_h^r\} \cup \{v_b^r\}$, i.e., modifying (only) the first difference between S' and S_M to be the same as S_M. The key idea is to modify S' step by step until S' becomes S_M, and in this procedure the *cost* is never increased.

Now, we will prove that $cost(S'') \leq cost(S')$. We use w_1 and w_2 to denote the biggest weight value in $V_h' \setminus \{v_b^r\}$ and $V_x' \setminus \{v_h^r\}$, respectively. $w_1 \geq w_2$ and $w(v_h^r) \geq w(v_b^r)$ since we descend the position of vertices in each clique according to the weight value. Thus, $cost(S') - cost(S'') = max\{w_1, w(v_b^r)\} + max\{w_2, w(v_h^r)\} - max\{w_1, w(v_h^r)\} - max\{w_2, w(v_b^r)\}$. There are 6 possibilities, which are divided into two cases as follows:

(1) The case $w_1 \geq w_2 \geq w(v_h^r) \geq w(v_b^r)$ and $w(v_h^r) \geq w(v_b^r) \geq w_1 \geq w_2$. Thus, $cost(S') - cost(S'') = 0$;

(2) The case $w_1 \geq w(v_h^r) \geq w_2 \geq w(v_b^r)$, $w(v_h^r) \geq w_1 \geq w_2 \geq w(v_b^r)$, $w_1 \geq w(v_h^r) \geq w(v_b^r) \geq w_2$ and $w(v_h^r) \geq w_1 \geq w(v_b^r) \geq w_2$. Thus, $cost(S') - cost(S'') \geq 0$.

According to (1) and (2), we find that $cost(S'') \leq cost(S')$. We repeat the above process (i.e., modifying the first difference between S'' and S_M) to improve the S'', and conclude that S'' will become the same as S_M. At last, $cost(S_M) = cost(S'') \leq cost(S')$. The hypothesis of $cost(S') < cost(S_M)$ is not valid.

Note that $cost(S_M)$ is the lower bound of $G' = (V', E')$ where $V' = C_1 \cup C_2 \ldots \cup C_t$. Thus, the WGCP's solution value of G must be at least $cost(S_M)$. □

The lower bound helps to prove the optimality for some instances. If the algorithm finds a solution whose cost meets the lower bound, then it is proved to be optimal.

Reduction Rule

In this subsection, we propose our reduction rule exploiting clique sampling, which is used to reduce the size of the original graph.

Reduction Rule: Given a vertex weighted graph $G = (V, E, w)$, maximal weighted cliques $CliSet = \{C_1, C_2, \ldots, C_t\}$ in it and $S_M = C_1 \otimes C_2 \otimes \ldots \otimes C_t = \{V_1, V_2, \ldots, V_l\}$. For $\forall u \in V$, let $k(u) = d(u) + 1$, if $k(u) \leq l$ and $w(u) < max_{v_j \in V_{k(u)}} w(v_j)$, the optimal cost of G is unchanged after removing vertex u from G.

Proof of the soundness of the rule: Let us consider a vertex u with $k(u) \leq l$. In the following, u is explicit from the context, and thus we simply use k to denote $k(u)$. We will prove that if $w(u) < max_{v_j \in V_k} w(v_j)$, then removing u from G does not have impact on the optimal cost of WGCP on G.

For convenience, we use v_j^{max} to denote the vertex with the biggest weight in V_j ($1 \leq j \leq l$), and if some vertices in V_j have the same biggest weight, then among them we randomly pick a vertex as v_j^{max}.

Since $k \leq l$, according to the definition of operation $\otimes$, there exists a weighted clique C in $CliSet$ such that $|C| \geq k$ and v_k^{max} is the k^{th} vertex in C. Also, the definition of operation $\otimes$ allows us to have the weights of the first k vertices in C not smaller than $w(v_k^{max})$. Here, we will prove that $u \notin C$. We have two cases. 1) $|C| > k$. Since C is a clique containing at least $k + 1$ vertices, the degree of each vertex

in it is at least k. But $d(u) = k - 1$, so $u \notin C$; 2) $|C| = k$. Because the weight value of any vertex in C is not smaller than $w(v_k^{max})$ and $w(u) < w(v_k^{max})$, $u \notin C$. Together, we know $u \notin C$. Thus, $C \subseteq V \setminus \{u\}$.

Suppose S' is an optimal WGCP coloring of the subgraph induced by $V \setminus \{u\}$. In the following, we will prove the optimal WGCP coloring of V is $cost(S')$. As this induced subgraph contains the clique C, the first k vertices in C are colored using k colors, and let us denote this color set as K. On the other hand, $N(u)$ is colored using at most $d(u) = k - 1$ colors. Therefore, S' is extended to an optimal WGCP coloring S^* of the original graph in this way: assign u with a color in K that is not assigned to any vertex of $N(u)$. Recall that the weights of the first k vertices in C are not smaller than $w(v_k^{max})$, which is larger than $w(u)$. Thus, for all colors in K, there is always at least one vertex whose weight is larger than $w(u)$, so the coloring of u does not increase the optimal cost, i.e., $cost(S') = cost(S^*)$. □

The proposed reduction rule is inspired by the reduction rule for the GCP problem (Lin et al. 2017), but has an essential difference. Our rule is based on sampling maximal weighted cliques, while the previous rule for GCP depends on only one maximal clique. Sophisticated techniques are required to re-arrange the sampled cliques in order to get the lower bound for WGCP.

Based on the proposed reduction rule, we introduce a reduction procedure called ReductionWGCP (Algorithm 2). We maintain two sets $RemoveSet$ and $OperateV$, which will be used in the local search algorithm.

- $RemoveSet$ stores the vertices removed by the reduction rule;
- $OperateV$ denotes the set of vertices for which the color assignment can be modified during the local search, as we can fix the color for a clique according to the symmetry of colorings. Specifically, $OperateV$ equals the vertex set V minus both $RemoveSet$ and maximum weighted clique C^* that is obtained in the clique sampling. The vertex to be operated is always selected from $OperateV$.

Algorithm 2: ReductionWGCP(G)

Input: a weighted graph $G = (V, E, w)$
Output: $RemoveSet$, $OperateV$ and a weighted clique C^*

1 $CliSet := FastWClq(G)$;
2 find a maximum weighted clique C^* from $CliSet$;
3 $S_M := C_1 \otimes C_2 \otimes \ldots \otimes C_t$, where $CliSet = \{C_1, \ldots, C_t\}$;
4 $LB := cost(S_M)$, $RemoveSet := \emptyset$;
5 **foreach** *vertex* v_i *in* $V \setminus C^*$ **do**
6 **if** v_i *can be removed by* **Reduction Rule then**
7 $RemoveSet := RemoveSet \cup \{v_i\}$;
8 $OperateV := V \setminus (C^* \cup RemoveSet)$;
9 **return** (LB, $RemoveSet$, $OperateV$, C^*);

Initially, the algorithm computes some maximal weighted cliques $CliSet$ (line 1), and the maximum weighted clique C^* is obtained among the cliques (line 2). The algorithm initializes the lower bound LB and removed set $RemoveSet$ (lines 3-4). In lines 5-7, according to reduction rule, the algorithm adds some removed vertices into $RemoveSet$. Subsequently, the algorithm initializes $OperateV$. Finally, LB, $RemoveSet$, $OperateV$ and C^* are returned (line 9).

Local Search Algorithm

This section presents the local search algorithm in our RedLS method. We first introduce the two main ideas in the algorithm and then describe the local search algorithm in detail.

Selection rules and candidate sets

When we find a better feasible coloring $S = \{V_1, \ldots, V_z\}$, we try to select $V_i \in S$ and move all vertices in $Max_v(V_i) = \{v_j | w(v_j) \geq w(v_t), v_j \in V_i, \forall v_t \in V_i\}$ to another color class V_j, and thus obtain a new infeasible coloring S'. In the above process, $Max_v(V_i)$ is denoted as some vertices in V_i with the biggest weight. The selected vertices follow the rules detailed below.

Selection Rule 1: select some vertices with the biggest $(cost(S) - cost(S')/(\sum_{v \in Max_v(V_i)} |score(v, V_i, V_j)|)$ value, where $1 \leq i, j \leq l$ and $i \neq j$.

Based on the above rule, we intend to construct a new S' with the smallest total weight of conflict edges and minimum weight of S'. When S is an infeasible coloring, we maintain three candidate operation sets $CanSet_1$, $CanSet_2$ and $CanSet_3$. During the local search process, we mainly use the above candidate sets and configuration checking which will be mentioned in the next subsection to modify the candidate solution. We assume that $S'' = S \oplus \langle v, V_i, V_j \rangle$ and S^* denotes the best found solution. The three sets are defined as follows.

$CanSet_1 = \{\langle v, V_i, V_j \rangle | score(v, V_i, V_j) > 0, cost(S'') < max\{cost(S), cost(S^*)\}\}$;

$CanSet_2 = \{\langle v, V_i, V_j \rangle | score(v, V_i, V_j) = 0, cost(S'') < cost(S)\}$;

$CanSet_3 = \{\langle v, V_i, V_z \rangle | score(v, V_i, V_z) > 0, cost(S'') < cost(S^*), z > |S^*|\}$.

It should be noted that V_j in $CanSet_1$ and $CanSet_2$ is a color class in S. During the local search, if the three above sets are empty sets, then we apply selection rule 2 to decide which operation should be selected.

Selection Rule 2: pick a random conflict edge $e_i \in CE(S)$. (1) We select $\langle v, V_i, V_j \rangle$ with the biggest $score$ value with $cost(S'') < cost(S^*)$, where $v \in e_i$ and $V_j \in S$. (2) If the operation does not exist, we randomly select $\langle v, V_i, V_j \rangle$ where $v \in e_i$, $V_j \in S$ and $i \neq j$.

CC for WGCP

We propose a variant of the configuration checking strategy (Cai, Su, and Sattar 2011) to deal with the cycling problem and denote this CC variant for WGCP as CC-WGCP.

We implement CC-WGCP with a Boolean array named $conf$ for vertices where $conf[v] = 1$ means v is allowed to move to a different color class, otherwise $conf[v] = 0$.

All $conf$ values are initialized as 1. With the $conf$ array, the CC-WGCP strategy is well described by the following rule:

CC-WGCP Rule After performing $\langle v, V_i, V_j\rangle$, (1) $conf[v]$ is set to 0, and (2), if $v \in CanSet_1$, then for $\forall u \in N(v)$, $conf[u]$ is set to 1;

Intuitions underlying the CC-WGCP rule are given below. Each local search step performs an operation $\langle v, V_i, V_j\rangle$, which moves vertex v from color class V_i to class V_j. After the operation, we set $conf[v] = 0$, which prevents v from changing its color again until $conf[v]$ is set to 1. Therefore, a key point is the condition to set $conf$ value to 1.

If the performed operation $\langle v, V_i, V_j\rangle \in CanSet_1$, then the execution of $\langle v, V_i, V_j\rangle$ reduces both $g(S)$ and $cost(S)$, which means that vertex v has been likely moved to a class that is more suitable for it. Considering the color assignment of a vertex has a direct impact on the permissible colors of its neighbors, once v is moved to the "right" set, it is reasonable to adjust the colors of its neighbors in the following steps. Thus, we encourage vertices in $N(v)$ to be moved by setting their $conf$ values to 1.

For operations in other candidate operation sets, we do not modify $conf$ values, as those operations can hardly improve the solution. It should not be encouraged to follow the consequences of such operations.

Description of RedLS algorithm

The main body of RedLS (Algorithm 3) is a local search algorithm (lines 2-24), and we present it in this section.

The algorithm first constructs an initial candidate solution (by the ConstructWGCP function) in line 2. ConstructWGCP builds an initial candidate solution S by first establishing a color class for each vertex of C^* (as C^* is a clique) and then iteratively putting each vertex in $OperateV$ into a proper color class without causing any conflict. If a vertex cannot be put into any existing color class without causing a conflict, then a new color class is established, and the vertex is put into it. In each iteration, a vertex in $OperateV$ is chosen to put into a color class using the BMS heuristic (Cai 2015). Specifically, the algorithm randomly selects t vertices and among them picks the vertex v with the biggest degree (in our work, t is set to 100). The algorithm randomly picks a color class $V_i \in S$ which does not contain any vertex adjacent to v. If no such color class exits, a new color class $V_{|S|+1}$ is created, and vertex v is then put into the new class. The complexity of the construction procedure ConstructWGCP is $O(|E|)$.

During the local search procedure (lines 3-24), when a better feasible coloring S is obtained, the algorithm iteratively picks a random $\langle v, V_i, V_j\rangle$ from $CanSet_2$ to modify S until $CanSet_2$ is empty (lines 5-6), which can further reduce the cost. S^* is updated by S (line 7). If the cost of S^* is equal to the lower bound LB, then the optimal solution is found and is returned (lines 8-9). All the $conf$ values should be reset to 1 (line 10). Based on selection rule 1, the algorithm updates S by moving some vertices (line 11).

Each iteration of the local search is described as follows. First, if $CanSet_1$ is not empty, then the algorithm selects $\langle v, V_i, V_j\rangle$ with $conf[v] = 1$ via the BMS heuristic (line 13). Subsequently, S is updated accordingly (line 14). $conf(v)$ is set to 0 and the algorithm sets $conf(u)$ to 1 for each $u \in N(v)$, which means that vertices in $N(v)$ can be moved in the following steps. Otherwise, the algorithm attempts to update S by performing any operation in $CanSet_2$ (lines 16-18). The algorithm selects a random operation in $CanSet_3$ as the next operation (lines 19-20). If $CanSet_3$ is empty, $w_e(e_i)$ is increased by one for each conflict edge $e_i \in CE(S)$ (line 22). The algorithm uses selection rule 2 to pick the next operation (line 23). Finally, S and the corresponding $conf$ are updated (line 24).

Algorithm 3: RedLS(G,*cutoff*)

Input: a weighted graph $G = (V, E, w)$, the *cutoff* time
Output: a feasible coloring $S^* = \{V_1^*, V_2^*, \ldots, V_z^*\}$of G

1 initialize LB, $RemoveSet$, $OperateV$ and C^* by ReductionWGCP(G);
2 $S^* := S :=$ConstructWGCP($OperateV$,C^*);
3 **while** *elapsed time* $<$ *cutoff* **do**
4 **if** $g(S) = 0 \&\& cost(S) < cost(S^*)$ **then**
5 **while** $CanSet_2 \neq \emptyset$ **do**
6 select a random operation $\langle v, V_i, V_j\rangle$ from $CanSet_2$, $S := S \oplus \langle v, V_i, V_j\rangle$;
7 $S^* = S$;
8 **if** $cost(S^*) = LB$ **then**
9 **break**;
10 reset $conf[v]$ to 1 for $\forall v \in V$;
11 select some operations based on **Selection Rule 1** and update S by performing the above operations;
12 **if** $CanSet_1 \neq \emptyset$ **then**
13 select $\langle v, V_i, V_j\rangle$ with the biggest $score$ value and $conf[v] = 1$ among t samples from $CanSet_1$;
14 $S := S \oplus \langle v, V_i, V_j\rangle$ and update the corresponding $conf$ according to **CC-WGCP Rule**;
15 **else**
16 **while** $CanSet_2 \neq \emptyset$ **do**
17 select $\langle v, V_i, V_j\rangle$ from $CanSet_2$ randomly;
18 $S := S \oplus \langle v, V_i, V_j\rangle$ and update $conf$ according to **CC-WGCP Rule**;
19 **if** $CanSet_3 \neq \emptyset$ **then**
20 select $\langle v, V_i, V_j\rangle$ from $CanSet_3$ randomly;
21 **else**
22 $w_e(e_i) := w_e(e_i) + 1$, for $\forall e_i \in CE(S)$;
23 select $\langle v, V_i, V_j\rangle$ based on **Selection Rule 2**;
24 $S := S \oplus \langle v, V_i, V_j\rangle$ and update the corresponding $conf$ according to **CC-WGCP Rule**;
25 put each vertex in $RemoveSet$ into a proper color class in S^*;
26 **return** S^*;

Table 1: Results of RedLS, AFISA, MWSS and 2Phase on conventional benchmarks.

Instance	RedLS MIN(AVG)	AFISA MIN(AVG)	MWSS MIN	2Phase MIN(AVG)	Instance	RedLS MIN(AVG)	AFISA MIN(AVG)	MWSS MIN	2Phase MIN(AVG)
COLOR benchamrk					COLOR benchamrk				
C2000.5	**2138(2184.8)**	2400(2425.1)	N/A	2983(3019.9)	le450_25b	**307(310.1)**	318(325.8)	489	346(364.75)
C2000.9	**5507(5553.9)**	6228(6284)	N/A	5799(5832.05)	le450_25c	**347(357.1)**	378(387.9)	606	464(476.1)
DSJC1000.1	**304(306.8)**	354(358.9)	N/A	374(564.5)	le450_25d	**335(343.7)**	375(385.3)	587	461(470.9)
DSJC1000.5	**1197(1220.9)**	1354(1371.3)	N/A	1525(1543.45)	miles1000	**430(432.8)**	432(444.7)	577	453(457)
DSJC1000.9	**2856(2887.1)**	3166(3231)	3759	2977(3007.25)	miles1500	797(797)	**587(644.3)**	868	800(802)
DSJC125.1g	**23(23.6)**	**23**(24)	25	24(24.8)	miles250	**102(102.4)**	**102**(102.7)	169	103(105.65)
DSJC125.1gb	**90(92.2)**	**90**(92.5)	95	95(97.35)	miles500	**260(260.9)**	**260**(261.3)	396	268(272.05)
DSJC125.5g	**71(72.2)**	**71**(72.3)	78	76(78.2)	mulsol.i.5	**367(367)**	**367(367)**	436	368(369.05)
DSJC125_5gb	**243(244.2)**	**243**(250.2)	263	251(255.9)	queen10_10	**162(165.2)**	166(169.2)	233	170(172.6)
DSJC250.1	**127(131.6)**	140(141.9)	227	174(181.65)	queen11_11	**174(178)**	178(182.3)	282	183(188.15)
DSJC250.5	**398(403.5)**	415(428.1)	575	427(435.75)	queen12_12	**187(190.8)**	194(198.6)	282	205(211.7)
DSJC250.9	936(939.5)	**925(942.7)**	1153	990(990.9)	queen13_13	**195(201.3)**	204(207.5)	299	218(225.4)
DSJC500.1	**187(191.7)**	210(215.6)	342	234(289.75)	queen14_14	**217(223.6)**	224(227.4)	351	238(249.3)
DSJC500.5	**707(716.2)**	778(845.1)	1086	808(834)	queen15_15	**225(233.4)**	237(241.2)	364	268(278)
DSJC500.9	**1670(1683)**	1790(1854.5)	2103	1744(1750.5)	queen16_16	**237(243.5)**	250(254.8)	380	291(305.4)
DSJR500.1	**169(174.6)**	**169**(175.4)	266	178(193.8)	R100_5gb	**220(222.4)**	221(224.1)	225	234(235.15)
flat1000_50_0	**1155(1180.1)**	1289(1315.7)	N/A	1481(1502.85)	wap01a	**545(594.1)**	638(653.1)	N/A	645(647.7)
flat1000_60_0	**1192(1219.6)**	1338(1354)	N/A	1525(1546.15)	wap02a	**538(571.1)**	637(638.1)	N/A	663(674.05)
flat1000_76_0	**1170(1197.7)**	1314(1337.6)	N/A	1504(1519.1)	wap03a	**563(581.4)**	687(707.5)	N/A	709(713.5)
inithx.i.1	**569(571.5)**	587(587.9)	N/A	623(637.05)	wap04a	**561(574.3)**	698(709)	N/A	718(724.4)
inithx.i.2	**329(337.9)**	341(341.6)	N/A	353(368.6)	wap05a	**542(545)**	598(610.9)	N/A	792(809.15)
inithx.i.3	**337(343.9)**	352(355.6)	N/A	347(370.15)	wap06a	**517(530.4)**	599(607.6)	N/A	631(661)
latin_square_10	**1532(1563.9)**	1690(1900)	2458	1610(1617.3)	wap07a	**555(564.4)**	680(692.5)	N/A	731(1106.4)
le450_15a	**212(218.8)**	241(247.1)	394	245(322.25)	wap08a	**534(542.7)**	663(673.4)	N/A	668(668.15)
le450_15b	**217(222)**	239(245.1)	380	321(333.1)	zeroin.i.1	**511(511.1)**	518(518)	786	519(519)
le450_15c	**284(290.4)**	313(320.8)	512	396(406.05)	zeroin.i.2	**336(336.1)**	**336**(337.6)	464	343(348.6)
le450_15d	**279(287.1)**	306(314.1)	518	381(400.9)	zeroin.i.3	**298(299.5)**	299(301.7)	483	313(321)
le450_25a	**306(307.5)**	317(329.9)	496	371(397.9)					
Matrix decomposition benchmarks					Matrix decomposition benchmarks				
r12	**7690(7706.7)**	7691(7710.4)	**7690***	8401(9031.35)	r19	**6826(6863.6)**	6840(6868.1)	**6826***	7778(8371.05)
r13	**7500(7525.3)**	7521(7558.3)	**7500***	8113(9132.45)	r25	**8426(8507.1)**	8468(8560.8)	**8426***	8928(10209.95)

After the time limit is reached, RedLS assigns each vertex in $RemoveSet$ into a proper color class in S^*, and this is performed very quickly according to the proof of the reduction rule (line 25). Finally, the algorithm returns S^*.

Experimental Evaluation

We evaluate RedLS on a broad range of massive graphs and conventional benchmarks, compared with three state-of-the-art algorithms, including an exact algorithm MWSS (Cornaz, Furini, and Malaguti 2017) as well as two heuristic algorithms, i.e., 2Phase (Malaguti, Monaci, and Toth 2009) and AFISA (Sun et al. 2018).

We considered 161 conventional benchmarks in (Sun et al. 2018), which are mainly divided into two parts: (1) COLOR benchmark [2]; (2) two matrix decomposition benchmarks (named rxx and pxx) (Prais and Ribeiro 2000). These WGCP benchmarks are originally weighted graphs.

As for large instances, we select 187 massive graphs from the Network Data Repository (Rossi and Ahmed 2015). For the sake of space, we do not report the results on graphs with fewer than 100,000 vertices or fewer than 1,000,000 edges. Hence, we select a total of 65 massive graphs. These graphs are originally unweighted, and we use two weighting functions. (1) We employ the same method as in (Cai and Lin 2016; Wang, Cai, and Yin 2017), i.e., for the ith vertex v_i, the weighting function $w_1(v_i)$=(i mod 200)+1; (2) Observed from the real weighting functions of error-correcting codes and winner determination problem (McCreesh et al. 2017), vertices with low degree have high weight values, while vertices with high degree have light weights. According to our experiments, the following weighting function can well simulate the weight distributions from those real world instances, and thus is adopted to generate weights. (a) if $d(v) \in [0, 0.35 \times d_{max}]$, then $w_2(v)$=8; (b) if $d(v) \in (0.35 \times d_{max}, 0.6 \times d_{max}]$, then $w_2(v)$=4; (c) if $d(v) \in (0.6 \times d_{max}, 0.85 \times d_{max}]$, then $w_2(v)$=2; (d) if $d(v) \in (0.85 \times d_{max}, d_{max}]$, then $w_2(v)$=1 , where $d_{max} = max\{|N(v)|\}$, for $\forall v \in V$.

RedLS and all competitors are implemented in C++ and compiled by g++ with '-O3' option. All experiments are run on Intel Xeon E5-2640 v4 @ 2.40GHz CPU with 128GB RAM under CentOS 7.5. All algorithms are executed 20 times on each instance independently with a cutoff time of 3600 s. The running time of RedLS includes two parts (i.e., reduction procedure and local search algorithm). For each instance, MIN denotes the weight of best solution found, and AVG denotes the average weight of the solution obtained in 20 runs. If an algorithm fails to provide a solution within the given time limit, then the corresponding column is marked as "N/A". If an algorithm proves the optimal solution, then the corresponding column is marked with a "$*$".

[2] https://mat.gsia.cmu.edu/COLOR02/

Table 2: Results of RedLS and AFISA on massive graphs with w_1.

Instance	RedLS MIN(AVG)	AFISA MIN(AVG)	Instance	RedLS MIN(AVG)	AFISA MIN(AVG)
bn-human-BNU_1_0	**31034**	40486	soc-delicious	**2248(3006.5)**	3260(3422.2)
025865_session_1-bg	**(31630)**	(41151)	soc-digg	**7218(9598.5)**	10899(11223.8)
bn-human-BNU_1_0	**21230**	27205	soc-dogster	**7728(8278.8)**	10856(11077.3)
025865_session_2-bg	**(21538.3)**	(27539.6)	soc-flickr	**12971(13266.9)**	16597(16872.9)
ca-coauthors-dblp	**37905***	**37905(37905)**	soc-flickr-und	**19525(20204.1)**	25345(25586.2)
ca-dblp-2012	**14115***	**14115(14115)**	soc-flixster	**4783(5473.4)**	N/A
ca-hollywood-2009	**222720***	N/A	soc-FourSquare	**3731(4073.6)**	4922(4999.9)
channel-500x100x 100-b050	**1387(1542.2)**	N/A	soc-lastfm	**2893(3472.9)**	4972(5186.9)
			soc-livejournal	**23102(23324.2)**	N/A
dbpedia-link	**7351(8755.5)**	N/A	soc-livejournal-user-groups	**6092(6256.8)**	N/A
delaunay_n22	**1201(1264.6)**	N/A	soc-LiveMocha	**3930(4014.6)**	6256(6353.7)
delaunay_n23	**1274(1320.2)**	N/A	soc-ljournal-2008	**40559(40561.6)**	N/A
delaunay_n24	**941(1109.5)**	N/A	soc-orkut	**11873(13225.2)**	N/A
friendster	**5850(5989.5)**	N/A	soc-orkut-dir	**12697(14326.5)**	N/A
hugebubbles-00020	**700(713.5)**	N/A	soc-pokec	**3871(5174.4)**	5226(5298.3)
hugetrace-00010	**674(694.4)**	N/A	soc-sinaweibo	**10165(11040.9)**	N/A
hugetrace-00020	**691(700.8)**	N/A	soc-twitter-higgs	**8822(9683.5)**	9533(9702.5)
inf-europe_osm	**759(759)**	N/A	soc-youtube	**3547(3873.4)**	5113(5237.6)
inf-germany_osm	**634(634.4)**	N/A	soc-youtube-snap	**3519(3981.4)**	5052(5129.8)
inf-roadNet-CA	**781(786.2)**	967(971.5)	socfb-A-anon	**3709(4773.2)**	N/A
inf-roadNet-PA	**767(855.2)**	955(955.5)	socfb-B-anon	**3269(4880.8)**	N/A
inf-road-usa	**788(788.2)**	N/A	socfb-uci-uni	**1090(1265.2)**	N/A
rec-dating	**5137(5235.9)**	9880(9491.1)	tech-as-skitter	**7249(7734.6)**	8863(9013.6)
rec-epinions	**2532(2900.8)**	4977(5097.1)	tech-ip	**833(940.4)**	N/A
rec-libimseti-dir	**4538(4908.2)**	8554(8291.7)	twitter_mpi	**20933(22374.8)**	N/A
rgg_n_2_23_s0	**2838(2916.3)**	N/A	web-arabic-2005	**12258(12258)**	**12258(12258)**
rgg_n_2_24_s0	**2965(3082.7)**	N/A	web-baidu-baike	**4703(5989.9)**	N/A
rt-retweet-crawl	**1702(2105.5)**	2489(2615.1)	web-it-2004	**46330(46330)**	**46330(46330)**
sc-ldoor	**5660(7035.3)**	7605(7645.5)	web-uk-2005	**54850***	**54850(54850)**
sc-msdoor	**5691(6864.9)**	7458(7501.8)	web-wikipedia_link	**90278(90427)**	N/A
sc-pwtk	**5651(6781.7)**	7120(7124.8)	web-wikipedia2009	**4377(4387.2)**	4384(4387.4)
sc-rel9	**1010(1091.7)**	N/A	web-wikipedia -growth	**6296(9268.9)**	N/A
sc-shipsec1	**4177(4728.5)**	4990(5021.7)			
sc-shipsec5	**4770(5331.9)**	5916(5917.3)			
soc-buzznet	**7031(7674.2)**	9994(10095.4)	wikipedia_link_en	5336(5433.1)	N/A

Results on conventional benchmarks

For all conventional benchmarks, RedLS finds better values than 2Phase for 96 instances. Thus, we mainly compare RedLS with MWSS and AFISA. Most instances are so easy that RedLS, MWSS and AFISA find the same quality values. We do not report the detailed results of such instances in Table 1, but we summarize the run time comparisons in Figure 1. Specifically, Figure 1 shows the average running time of RedLS and the corresponding competitor when both algorithms find the same minimum solution values, clearly showing the superiority of RedLS, with a few exceptions.

For the remaining 59 instances, which are more difficult, the results are reported in Table 1. For all 59 instances, RedLS performs better than 2Phase. Moreover, among the 59 instances, RedLS outperforms AFISA and MWSS on 48 and 55 of them, respectively. Compared to AFISA, RedLS is worse in only 2 instances. Additionally, RedLS proves the optimal solution for 4 instances from the conventional benchmarks, and the number of reduced vertices is on average 12.45, indicating that the reduction rule is not effective for conventional benchmarks. The good results of RedLS on conventional benchmarks mainly come from the power of the underlying local search algorithm.

Results on massive graphs with w_1 function

Note that MWSS fails to find a solution for many of the conventional graphs and all of the massive graphs, mainly due to its memory-expensive data structure (i.e., MWSS stores a auxiliary graph whose size is larger than the size of complementary graph), while 2Phase fails to obtain a solution for all of the massive graphs, mainly due to high-complexity heuristics (i.e., the first phase of 2Phase produces a very large number of independent sets, which wastes lots of time). Thus, we mainly report the results of RedLS and AFISA on Table 2. For all the 65 massive instances, RedLS finds better or same-quality solutions than AFISA.

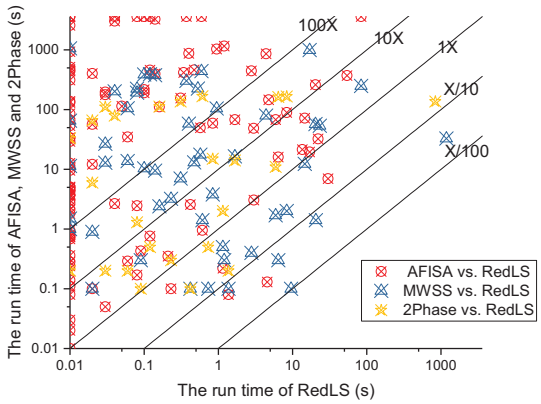


Figure 1: Average running time of RedLS and competitors.

Specifically, RedLS obtains better solutions on 60 instances. For the remaining 5 instances, RedLS and AFISA both find the same solutions. Furthermore, RedLS proves the optimal solution for 4 instances. Figure 2 shows the percentage of reduced vertices after applying the reduction rule. The reduction rule removes on average 60.59% vertices of all massive instances. For 42 massive instances, the percentage of reduced vertices exceeds 50% and this significantly demonstrates the effectiveness of the reduction rule.

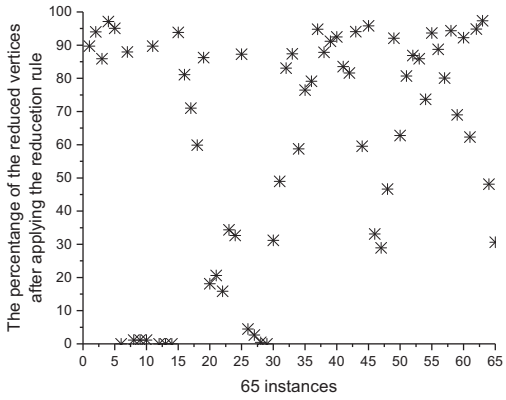


Figure 2: The percentage of the reduced vertices.

Results on massive graphs with w_2 function

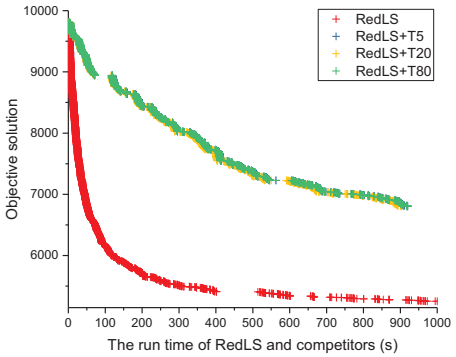
Table 3 shows a summary on the comparisons between RedLS and three competitors (i.e., AFISA, MWSS and 2Phase). Once again, RedLS outperforms three competitors on all massive graphs with w_2. This indicates that RedLS can greatly improve the solution quality in massive graphs. Surprisingly, RedLS proves the optimal solution for 19 instances.

Table 3: Summary of comparison between RedLS, AFISA, MWSS and 2Phase on massive graph with w_2. #Better indicates the number of instances where an algorithm finds better minimal (average) solutions. #N/A denotes the number of instances where an algorithm fails to find a solution under the given time limit.

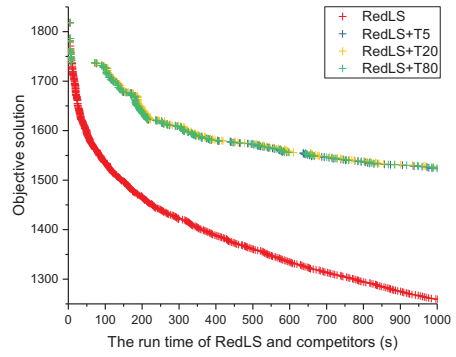
Benchmark		RedLS	AFISA	MWSS	2Phase
massive graph	#Better	**60(60)**	0(0)	0(0)	0(0)
with w_2	#N/A	0	32	65	65

The Effectiveness of the CC-WGCP Strategy

To verify the effectiveness of the CC-WGCP strategy, we use tabu mechanism (Glover 1989; Sun et al. 2018) instead of the CC-WGCP strategy and design three alternative algorithms: RedLS+T5, RedLS+T20, and RedLS+T80 where the tabu tenure tt is set to 5, 20, and 80, respectively. During the local search procedure, the tabu mechanism will prevent the search from revisiting the selected vertex for the next tt iterations. Figure 3 shows that RedLS reaches better solution values than three competitors within the same time limit on two selected instances. This indicates that the CC-WGCP



(a) Results on instance rec-dating



(b) Results on instance sc-rel9

Figure 3: Evolution of objective values with the run rime of RedLS and competitors on two selected instances.

plays a key role in the RedLS algorithm.

Conclusion

This paper introduced a lower bound, a reduction rule, and a local search algorithm for WGCP. We proposed the reduction rule based on clique sampling to remove some unnecessary vertices. In the local search algorithm, we designed the selection rules and the new variant of configuration checking to determine which operation is the candidate selected operation in the local search procedure. Experiments on conventional benchmarks and massive graphs indicate that RedLS significantly outperforms the state-of-the-art algorithms. As for future work, we will attempt to further improve RedLS via a few novel reduction rules.

Acknowledgments

This work is supported by the Fundamental Research Funds for the Central Universities 2412018ZD017, NSFC (under grant nos. 61806050, 61972063, 61976050, 61972384). Shaowei Cai was supported by Youth Innovation Promotion Association, Chinese Academy of Sciences (No.2017150).

References

Cai, S., and Lin, J. 2016. Fast solving maximum weight clique problem in massive graphs. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 568–574.

Cai, S.; Su, K.; and Sattar, A. 2011. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence* 175(9-10):1672–1696.

Cai, S. 2015. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, 747–753.

Cornaz, D.; Furini, F.; and Malaguti, E. 2017. Solving vertex coloring problems as maximum weight stable set problems. *Discrete Applied Mathematics* 217:151–162.

Furini, F., and Malaguti, E. 2012. Exact weighted vertex coloring via branch-and-price. *Discrete Optimization* 9(2):130–136.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Gavranovic, H., and Finke, G. 2000. Graph partitioning and set covering for the optimal design of a production system in the metal industry. *IFAC Proceedings Volumes* 33(17):603–608.

Glover, F. 1989. Tabu search-part i. *ORSA Journal on computing* 1(3):190–206.

Hébrard, E., and Katsirelos, G. 2018. Clause learning and new bounds for graph coloring. In *International Conference on Principles and Practice of Constraint Programming*, 179–194.

Hébrard, E., and Katsirelos, G. 2019. A hybrid approach for exact coloring of massive graphs. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 374–390.

Hochbaum, D. S., and Landy, D. 1997. Scheduling semiconductor burn-in operations to minimize total flowtime. *Operations research* 45(6):874–885.

Hsu, H.-C., and Chang, G. J. 2016. Max-coloring of vertex-weighted graphs. *Graphs and Combinatorics* 32(1):191–198.

Lin, J.; Cai, S.; Luo, C.; and Su, K. 2017. A reduction based method for coloring very large graphs. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 517–523.

Malaguti, E.; Monaci, M.; and Toth, P. 2009. Models and heuristic algorithms for a weighted vertex coloring problem. *Journal of Heuristics* 15(5):503–526.

McCreesh, C.; Prosser, P.; Simpson, K.; and Trimble, J. 2017. On maximum weight clique algorithms, and how they are evaluated. In *International Conference on Principles and Practice of Constraint Programming*, 206–225.

Pemmaraju, S. V.; Raman, R.; and Varadarajan, K. 2004. Buffer minimization using max-coloring. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, 562–571.

Peng, Y.; Choi, B.; He, B.; Zhou, S.; Xu, R.; and Yu, X. 2016. Vcolor: A practical vertex-cut based approach for coloring large graphs. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, 97–108.

Prais, M., and Ribeiro, C. C. 2000. Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment. *INFORMS Journal on Computing* 12(3):164–176.

Ribeiro, C. C.; Minoux, M.; and Penna, M. C. 1989. An optimal column-generation-with-ranking algorithm for very large scale set partitioning problems in traffic assignment. *European Journal of Operational Research* 41(2):232–239.

Rossi, R. A., and Ahmed, N. K. 2015. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 4292–4293.

Sun, W.; Hao, J.-K.; Lai, X.; and Wu, Q. 2018. Adaptive feasible and infeasible tabu search for weighted vertex coloring. *Information Sciences* 466:203–219.

Verma, A.; Buchanan, A.; and Butenko, S. 2015. Solving the maximum clique and vertex coloring problems on very large sparse networks. *INFORMS Journal on computing* 27(1):164–177.

Wang, Y.; Cai, S.; Chen, J.; and Yin, M. 2018. A fast local search algorithm for minimum weight dominating set problem on massive graphs. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 1514–1522.

Wang, Y.; Cai, S.; and Yin, M. 2016. Two efficient local search algorithms for maximum weight clique problem. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 805–811.

Wang, Y.; Cai, S.; and Yin, M. 2017. Local search for minimum weight dominating set with two-level configuration checking and frequency based scoring function. *Journal of Artificial Intelligence Research* 58:267–295.

Zhou, Y.; Duval, B.; and Hao, J.-K. 2018. Improving probability learning based local search for graph coloring. *Applied Soft Computing* 65:542–553.

Zuckerman, D. 2006. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, 681–690.