A Reduction based Method for Coloring Very Large Graphs

Jinkun Lin[1,2], **Shaowei Cai**[3,*], **Chuan Luo**[4] and **Kaile Su**[2,5]
[1]School of Electronics Engineering and Computer Science, Peking University, Beijing China
[2]College of Information Science and Technology, Jinan University, China
[3]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
[4]Institute of Computing Technology, Chinese Academy of Sciences, China
[5]Institute for Integrated and Intelligent Systems, Griffith University, Australia
jkunlin@gmail.com; shaoweicai.cs@gmail.com; chuanluosaber@gmail.com; k.su@griffith.edu.au;

Abstract

The graph coloring problem (GCP) is one of the most studied NP hard problems and has numerous applications. Despite the practical importance of GCP, there are limited works in solving GCP for very large graphs. This paper explores techniques for solving GCP on very large real world graphs. We first propose a reduction rule for GCP, which is based on a novel concept called degree bounded independent set. The rule is iteratively executed by interleaving between lower bound computation and graph reduction. Based on this rule, we develop a novel method called FastColor, which also exploits fast clique and coloring heuristics. We carry out experiments to compare our method FastColor with two best algorithms for coloring large graphs we could find. Experiments on a broad range of real world large graphs show the superiority of our method. Additionally, our method maintains both upper bound and lower bound on the optimal solution, and thus it proves an optimal solution when the upper bound meets the lower bound. In our experiments, it proves the optimal solution for 97 out of 144 instances.

1 Introduction

The graph coloring problem (GCP), also known as vertex coloring problem, requires to find an assignment of colors to vertices of a graph such that no two adjacent vertices share the same color while minimizing the number of colors. GCP is a fundamental combinatorial optimization problem and is NP-hard [Garey and Johnson, 1979], even to approximate within $n^{1-\epsilon}$ [Zuckerman, 2007]. It has been extensively studied not only for its theoretical aspects and for its difficulty, but also for its applications in many fields, including scheduling [Leighton, 1979], timetabling [de Werra, 1985], register allocation [Chow and Hennessy, 1990], and more recently to human subjects [Kearns *et al.*, 2006], among many others.

Recent advances in information technology, as well as the rapid growth of the Internet, have resulted in very large scale data sets. Many data sets can be modeled as graphs, and the study of massive real world graphs, also called complex networks, grew enormously in last decade. Many real world graphs of interest are very large (e.g., with tens of millions of vertices), and sparse, and the vertex degrees usually follow a power-law distribution [Newman, 2003]. Nevertheless, GCP remains hard to approximate when restricted to power-law graphs, unless $NP = ZPP$ [Shen *et al.*, 2012].

Despite its practical importance, there is limited research on solving GCP in massive graphs. Most literature devoted to solving it focuses on small graphs with up to thousands of vertices [Brélaz, 1979] [Campêlo *et al.*, 2008] [Hansen *et al.*, 2009] [Malaguti *et al.*, 2011] [Gualandi and Malucelli, 2012] [Hao and Wu, 2012].

Many existing algorithms for GCP become futile on massive graphs, due to their high space complexity and time complexity. For example, most GCP algorithms heavily rely on an adjacency matrix representation of the graph. Graphs with millions of vertices can not fit into a computer's working memory using this representation. Also, most commonly used strategies do not have sufficiently low time complexity, which severely limits their ability to handle massive graphs.

There has not been research on solving GCP in massive graphs until recent years. Rossi et al. proposed a method for coloring complex networks, which leverages triangles, triangle-cores and other properties and their combinations [Rossi and Ahmed, 2014]. Verma et al. exploited the k-core concept [Seidman, 1983] and developed an algorithm for GCP by successively coloring k-cores with decreasing k values [Verma *et al.*, 2015]. Peng et al. proposed a vertex-cut based method which partitions a graph into connected components and color them respectively. [Peng *et al.*, 2016].

In this work, we propose a novel method for solving GCP on massive sparse graphs. The method is based on a key concept called degree bounded independent set. An important observation is that we can reduce the graph by removing an ℓ-degree bounded independent set (ℓ is the lower bound of the chromatic number) while preserving optimal solutions. That is, any optimal solution to the remained graph can be extended to an optimal solution to the original graph by coloring the removed vertices iteratively. To improve the efficiency, we propose a heuristic algorithm for finding a high-quality clique, which serves as a lower bound; also, for improving the upper bound, we propose a heuristic algorithm

*Corresponding author

to color the whole graph after each round of reduction. We implement our method and it is named FastColor.

We carry out extensive experiments to evaluate the performance of FastColor on massive graphs, including real world graphs from various application fields. Experimental results show that, the solutions obtained by our method in quite short time limit (i.e., one minute) are nearly optimal and provably optimal for most instances. Particularly, for 97 out of 144 tested graphs, FastColor finds and proves an optimal solution in one minute. When compared with state of the art algorithms for GCP in massive graphs, FastColor also shows its superiority by finding better solutions, using much less time.

In the next section, we introduce some background knowledge. Then, we introduce a reduction rule for GCP based on degree bounded independent set in Section 3. After that, we describe the top-level algorithm of our method in Section 4, and its important components in Section 5. Experimental evaluations are presented in Section 6.

2 Preliminaries

Let G=(V,E) be an undirected graph where V=$\{v_1, v_2, \ldots, v_n\}$ is the set of vertices and E is the set of edges. Each edge is a 2-element subset of V. For a vertex v, its neighborhood is $N(v)$=$\{u \in V | \{u, v\} \in E\}$, and its degree is $d(v) = |N(v)|$. An edge e is called an *incident edge* of a vertex v iff $v \in e$. For a subset $V' \subseteq V$, we let $G[V']$ denote the subgraph induced by V', which is formed from V' and all of the edges connecting pairs of vertices in V'. Also, for a vertex $v \in V'$, we define $d_{G[V']}(v) = |N(v) \cap V'|$.

For a graph, a *proper coloring* is an assignment α of colors to all vertices of the graph such that no two adjacent vertices share the same color, and we say such a coloring colors the graph properly. We use $color(\alpha)$ to denote the set of colors used in a coloring α, and thus the number of colors used is $|color(\alpha)|$. The chromatic number of G, denoted as $\chi(G)$, is the smallest number of colors needed to color G properly. For a vertex v under α, the color assigned to it is $colorValue(v)$.

Given a graph G, a clique C is a set of pairwise adjacent vertices, while an independent set I is a set of pairwise non-adjacent vertices. A clique or independent set is maximal if it is not included in a larger clique or independent set. The clique number of a graph G, denoted as $\omega(G)$, is the number of vertices in the largest clique. We have $\chi(G) \geq \omega(G)$.

3 A Reduction Rule for Coloring

In this section, we introduce a reduction technique for GCP. Generally speaking, for a graph G, the idea is to decompose the graph into two parts, with the help of a lower bound on the chromatic number $\chi(G)$. For convenience, let us call one of them *kernel* while the other *margin*. We then reduce the graph by removing the margin, and seek for a proper coloring for the kernel.[1] Our reduction rule grantees that, any optimal solution for the kernel can be extended into an optimal solution for the original graph by coloring the removed margin iteratively. The power of our method also relies on the fact that, the reduction can be executed iteratively (i.e., a kernel can be taken as a new graph and be reduced again), while preserving the optimal solutions.

[1]The removed vertices and incident edges are stored in some data structure, so that they can be colored after the kernel is colored.

3.1 Reduction based on BIS

The proposed reduction rule for GCP is based on a concept called *degree bounded independent set*, which is formally defined as follows.

Definition 1 *Given a graph $G = (V, E)$, a k-degree bounded independent set is an independent set I s.t. $\forall v \in I$, $d(v) < k$.*

With the above definition, we propose a reduction rule denoted as BIS-Rule, where BIS stands for Bounded Independent Set. Additionally, we prove an important property about the rule.

BIS-Rule: *Given a graph $G = (V, E)$ and a lower bound on $\chi(G)$, denoted as ℓ, find an ℓ-degree bounded independent set I, and remove all vertices in I and their incident edges from G.*

Note that the rule depends on a parameter ℓ, which is a lower bound on $\chi(G)$.

Proposition 1 *Given a graph $G = (V, E)$ and an ℓ-degree bounded independent set I in it, and $\chi(G) \geq \ell$,*

1) if $\chi(G[V \setminus I]) < \ell$, then $\chi(G) = \ell$.

2) if $\chi(G[V \setminus I]) \geq \ell$, then $\chi(G) = \chi(G[V \setminus I])$.

Proof: Let us denote $G' = G[V \setminus I]$. Since I is an independent set, $\forall v \in I$, $N(v) \subseteq V(G')$.

1) The case $\chi(G') < \ell$. For any vertex $v \in I$, suppose under an optimal coloring to G', vertices in $N(v)$ are assigned with colors c_1, c_2,..., $c_{d(v)}$, where each $c_i \in \{1, 2, ..., \chi(G')\}$ and the values of c_i are not necessarily different to each other. Let $C_{N(v)}$ denote the set of colors assigned to $N(v)$. As $\chi(G') + 1 \notin C_{N(v)}$, we can assign color $\chi(G') + 1$ to vertex v without causing any conflict. Similarly, all other vertices in the independent set I can be assigned with color $\chi(G') + 1$. In this way, we obtain a proper coloring for graph G, using $\chi(G') + 1$ colors. Thus, we have

$$\chi(G) \leq \chi(G') + 1 < \ell + 1.$$

On the other hand, $\chi(G) \geq \ell$. Put them together, we have $\chi(G) = \ell$, and the coloring for G constructed in this way is optimal.

2) The case $\chi(G') \geq \ell$. For any vertex $v \in I$, suppose under an optimal coloring to G', $C_{N(v)}$ is the set of colors assigned to $N(v)$. (a) $C_{N(v)} \subseteq \{1, 2, ..., \chi(G')\}$ and $|C_{N(v)}| \leq d(v)$. (b) As I is ℓ-bounded, $\forall v \in I$, $d(v) < \ell$. (c) The assumption states $\chi(G') \geq \ell$. Put (a), (b), (c) together, we have

$$|C_{N(v)}| \leq d(v) < \ell \leq \chi(G').$$

So, there exists a color $c' \in \{1, 2, ..., \chi(G')\}$ s.t. $c' \notin C_{N(v)}$. We can assign this color c' to vertex v without causing any conflict. Similarly, all other vertices in I can be assigned with a color in $\{1, 2, ..., \chi(G')\}$ safely. In this way, we obtain a proper coloring for G, using $\chi(G')$ colors. Thus, we have $\chi(G) \leq \chi(G')$. On the other hand, since G' is a subgraph of G, $\chi(G) \geq \chi(G')$. Therefore, $\chi(G) = \chi(G')$, and the coloring for G constructed in this way is optimal. □

Proposition 1 shows that the BIS-Rule is sound, that is, we can always construct an optimal coloring for the original graph G from an optimal coloring for a graph G' which is reduced from G by the BIS-Rule. In our proof, we also show the construction method, which is very simple. When the found coloring for G' is not proved to be optimal, this construction method obtains a heuristic coloring for G.

We would like to note that, our reduction technique is essentially different from the previous 'Independent Set Extraction' method [Wu and Hao, 2012], which preprocesses the graph by iteratively removing an independent set and assigning a color to it. The 'Independent Set Extraction' method does not grantee to reserve the optimal solution after extracting an independent set. Indeed, for large graphs, many largest (extracted) independent sets are not part of an optimal coloring [Galinier *et al.*, 2013]. In this case, removing these independent sets will prevent inevitably the subsequent coloring algorithm from reaching an optimal coloring.

3.2 On Lower Bound Used in Reduction

As we have shown, the BIS-Rule exploits a lower bound on the chromatic number . Thus, before each round of reduction, we need to calculate a lower bound for the remained graph.

It is important to note that, any lower bound for any subgraph of G is a lower bound on $\chi(G)$ for the original input graph G. This is formally expressed in the following lemma.

Lemma 1 *Given a graph $G = (V, E)$, for any subgraph $G' = (V', E')$ of G, that is, $V' \subseteq V$ or $E' \subseteq E$ (or both), then $\chi(G) \geq \chi(G')$.*

The proof is trivial by contradiction. If there is a proper coloring, say α, with less than $\chi(G')$ colors for G, the projection on V' of α is also a proper coloring of G'. □

In our method, we find a lower bound of the chromatic number via finding a clique, as the clique number is a lower bound of the chromatic number. Although simple, clique-based lower bound is particularly effective in our reduction-based method. To see this, suppose the best found clique before some round of reduction is of size k, then the vertices removed by BIS-Rule are all of degree less than k. Therefore, any clique with size larger than k is reserved in the graph, while at the same time, the graph is reduced to a smaller size, which makes finding a larger clique more easily. Hence, as the algorithm processes, we can expect to find larger cliques, so that the lower bound on $\chi(G)$ is likely refined.

4 The Main Algorithm: FastColor

In this section, we introduce the top-level algorithm of our method named FastColor. Before going to detailed descriptions of the algorithm, we first introduce some notation.

G is the original graph, while G_k is the working graph which is reduced from G and becomes smaller during the algorithm. G_m collects all the vertices that have been removed from G, and removed edges are also stored accordingly (although not reflected in pseudo code). Notation lb_k denotes the lower bound on $\chi(G_k)$, while lb^* and ub^* denote the best found lower bound and upper bound on $\chi(G)$ respectively, and α^* denotes the best coloring found so far.

Algorithm 1: FastColor (G)

```
   Input: a graph G = (V, E)
   Output: A coloring assignment of G
1  G_k := G;
2  lb* := 0, ub* := |V|, α* = ∅;
3  lb_k := 0, isColored = false;
4  while elapsed time < cutoff do
5      lb_k := FindClq(G_k, lb_k);
6      if lb_k > lb* then lb* := lb_k;
7      I := find a maximal lb_k-degree bounded independent set;
8      G_k := remove I from G_k according to BIS-Rule;
9      G_m := G_m ∪ I;
10     if I ≠ ∅ then
11         lb_k := 0, isColored := false
12     α := ColorKernel(G_k, isColored);
13     α+ := a proper coloring for G extended from α by
           coloring vertices in G_m;
14     isColored := true;
15     if |color(α+)| < ub* then
16         α* := α+, ub* := |color(α+)|
17     if ub* := lb* then return α*;
18 return α*;
```

In the beginning, the working graph G_k is initialized as G; lb^*, ub^* and α^* are also initialized. After the initialization, a loop (lines 4-17) is executed until an optimal solution is proved (line 17) or a given time limit is reached. FastColor returns the best found coloring α^* for G upon reaching the termination condition (line 18).

Each iteration of the loop can be seen as three phases:

Lower bound computation (lines 5-6): The lower bound lb_k is computed by finding a clique in G_k. Since lb_k can also serve as a lower bound on $\chi(G)$, if lb_k is smaller (thus tighter) than lb^*, lb^* is updated accordingly.

Graph reduction (lines 7-9): To reduce the graph, we first find a maximal lb_k-degree bounded independent set. This is accomplished by traversing G_k sequentially and adding the vertex if its degree is less than lb_k and it is not adjacent to any vertex already in the independent set. Then, G_k is reduced by removing I, according to the BIS-Rule. Along with this reduction, removed vertices (and the removed incident edges) are stored into G_m. Note that sometimes the BIS-Rule cannot remove any vertex, and in this case, G_k is unchanged.

Graph coloring (lines 12-16): after the reduction, the original graph G is colored in two steps. First, the working graph G_k is colored by a coloring algorithm named $ColorKernel$. Then, the obtained coloring α for G_k is extended to a coloring α^+ for G by coloring G_m. This is accomplished iteratively, i.e., in each iteration the most recently removed uncolored independent set is colored, using a construction method as shown in the proof of Proposition 1. Finally, if the number of colors in α^+ is less than ub^*, then ub^* is updated accordingly.

Additionally, if ub^* meets lb^*, then a proved optimal coloring is found and returned (line 17).

5 Important Functions

In this section, we introduce two important functions (indeed sub-algorithms) in FastColor, one for finding clique before each round of reduction, while the other for coloring the remained graph after each round of reduction.

5.1 The FindClq Algorithm

We employ a construct-and-cut heuristic method [Cai and Lin, 2016] to find a high-quality clique from the remained graph. This algorithm, named $FindClq$ (Algorithm 2), can be viewed as a series of clique samples from the graph.

We use C to denote the current clique under construction, and $StartSet$ is the set containing vertices candidate as a starting vertex to construct a clique. $CandSet = \{v | v \in N(u)\ for\ \forall u \in C\}$ consists of candidate vertices eligible to extend the current clique.

Algorithm 2: FindClq (G, lb)

Input: A graph $G = (V, E)$, lower bound of clique size lb
Output: The size of best found clique

1 $lb_{old} := lb$;
2 $StartSet :=$ a set of random vertices from V ;
3 **while** $StartSet \neq \emptyset$ **do**
4 $u :=$ pop a random vertex from $StartSet$;
5 $C := \{u\}$;
6 $CandSet := N(u)$;
7 **while** $CandSet \neq \emptyset$ **do**
8 $v :=$ a vertex with greatest $|N(v) \cap CandSet|$ value among t samples from $CandSet$;
9 **if** $|C| + 1 + |N(v) \cap CandSet| \leq lb$ **then** $Break$;
10 $C := C \cup \{v\}$;
11 $CandSet := CandSet \setminus \{v\}$;
12 $CandSet := CandSet \cap N(v)$;
13 **if** $|C| > lb$ **then** $lb := |C|$;
14 **if** $lb_{old} = lb$ **then** adjust BMS parameter t;
15 **return** lb;

The $FindClq$ algorithm employs the BMS heuristic [Cai, 2015] in choosing the vertex to be added to the current clique. BMS heuristic returns the best element w.r.t. some comparison function among t samples (with replacement) from a given set. Since different t values correspond to different levels of greediness, we try several t values in $[1, t_{max}]$, where t_{max} is a parameter that needs to be specified, and is set to 64 in our experiments. We use a formula $t := 2t$ to get the next t value when $FindClq$ fails to find a larger clique after trying each vertex from $StartSet$ as the starting vertex (line 14). Also, when t exceeds t_{max}, it is reset to 1.

First, the algorithm chooses some random vertices from V to initialize the $StartSet$ (line 2). In our algorithm, the size of $StartSet$ is set to $\frac{|V|}{100}$. Then, the algorithm executes a loop until $StartSet$ becomes empty (lines 3-13), each iteration of which constructs a clique from a random vertex u popped from $StartSet$ (line 5). Along with adding the starting vertex u, $CandSet$ is initialized as $N(u)$ (line 6). Then, the clique is extended by iteratively adding a vertex v with the greatest value of $|N(v) \cap CandSet|$ among t samples from $CandSet$, until $CandSet$ becomes empty (lines 7-12).

Also, we use a cost-effective upper bound to prune the procedure (lines 9). Obviously, $|C| + 1 + |N(v) \cap CandSet|$ is an upper bound on size of any clique extended from C by adding v and more vertices.

At the end of a clique construction procedure, the lower bound of clique size lb is updated accordingly (line 13). Finally, when $FindClq$ terminates, it returns lb (line 14), which is used as a lower bound on the chromatic number.

5.2 The ColorKernel Algorithm

To color the kernel, we employ the concepts of *k-core* [Seidman, 1983] and *saturation degree* [Brélaz, 1979]

Definition 2 *Given a graph $G = (V, E)$ and a subset of vertices $V' \subseteq V$, a subgraph $G[V']$ is called a k-core if $d_{G[V']}(v) \geq k$ for $\forall v \in V'$.*

Definition 3 *Given a graph $G = (V, E)$ and a partial coloring assignment α, the saturation degree of a vertex is defined as the number of different colors used by $N(v)$ under α.*

If the graph G has not been colored (after each successful reduction, see Algorithm 1), we sort V according to core decomposition of G [Batagelj and Zaversnik, 2003] (line 3). This partitions V into smaller parts based on k-cores, and vertex in k-core with larger k has a larger index in V. Then we color V in decreasing array-index order (lines 4-7). Each vertex is colored with a minimum possible color.

Otherwise, we color V in an order depending on saturation degree. In each iteration, a vertex v with maximum saturation degree (breaking tie randomly) is selected (line 10) and is colored with a minimum possible color (line 12). To accelerate the selection operation, we use a bucket for each saturation degree to store the vertices of that saturation degree.

Algorithm 3: ColorKernel (G, $isColored$)

Input: a graph $G = (V, E)$, $isColored$
Output: a coloring assignment of G

1 $\alpha := \emptyset$;
2 **if** $isColored = false$ **then**
3 sort V according to core decomposition of G;
4 **for** *each $v \in V$ in decreasing array-index order* **do**
5 $c := min\{i > 0 \mid \forall u \in N(v), colorValue(u) \neq i\}$;
6 **if** $c > |color(\alpha)|$ **then** $c :=$recolor(v) ;
7 $\alpha := \alpha \cup (v, c)$;
8 **else**
9 **while** $V \neq \emptyset$ **do**
10 $v :=$a vertex from V with maximum saturation degree;
11 $V := V \setminus v$;
12 $c := min\{i > 0 \mid \forall u \in N(v), colorValue(u) \neq i\}$;
13 **if** $c > |color(\alpha)|$ **then** $c :=$recolor(v) ;
14 $\alpha := \alpha \cup (v, c)$;
15 update saturation degree accordingly;
16 **return** α;

During the coloring of a vertex v, if the minimum possible color is a new color, we use the *recolor* procedure [Rossi and

Ahmed, 2014] to avoid increasing the color number (lines 6, 13). It tries to change the color of v's neighbors, so that v can be colored with an existing color. Tomita et al. also use this technique in their MCS algorithm [Tomita *et al.*, 2010].

6 Experimental Evaluation

We conduct experiments on a broad range of real-world massive graphs to compare FastColor with two state of the art GCP algorithms proposed in [Rossi and Ahmed, 2014] and [Verma *et al.*, 2015]. These algorithms do not have names in the literatures, and are referred to as Rossi's algorithm and Verma's algorithm for convenience. We also like to compare FastColor with previous 'Independent Set Extraction' method [Wu and Hao, 2012], but the results on massive graphs are not available to us.

6.1 Experimental Preliminaries

FastColor is implemented in C++ and compiled with g++ version 4.8.4 with -O3 option. The experiments are carried out on a workstation under Ubuntu 14.04, using 2 cores of Intel i7-4710MQ CPU @ 2.50 GHz and 16 GB RAM.

We run FastColor 10 times on each graph, with a cutoff time of 60 seconds per run. For each graph, we report the minimum number of colors ("Min") found by FastColor, and the average number of colors over all runs ("Avg") if a 100 percent success rate is not reached. Besides, we also report the average runtime ("Time") of FastColor over all runs, where the runtime of an execution is the time it needs to find and prove the optimal solution if it proves the optimality, and the time to find the best coloring assignment otherwise. The number of reduction iterations ("#Iter.") is also shown.

Since the source code or binary of Rossi's and Verma's algorithm are not available to us, we compared FastColor with them using the results ("Min" and "Time") reported in the corresponding papers. The experimental environment of Rossi's algorithm is not reported in [Rossi and Ahmed, 2014], while Verma's algorithm was run on a workstation under windows 7, with two Intel E5620 CPU @ 2.40 GHz and 12 GB RAM. Despite the difference of platforms, we can still draw a conclusion clearly from the comparison that FastColor outperforms Rossi's and Verma's algorithms.

6.2 Results on Network Data Repository

In this subsection, we compare FastColor with Rossi's algorithm on the benchmarks from Network Data Repository online [Rossi and Ahmed, 2015].[2] Rossi's algorithm has several variants [Rossi and Ahmed, 2014], and we compare FastColor with the best one, i.e., TCORE-VOL with recolor procedure. As the runtime of TCORE-VOL *with* recolor is missing, we instead report the runtime of TCORE-VOL *without* recolor procedure (which seems to be shorter than it should be). The results of this variant on some instances are not reported in the literature, and we take the best solution we can find in [Rossi and Ahmed, 2014] and mark the runtime as '-'.

The results are presented in Tables 1 and 2. For all the 91 instances, FastColor obtains better or same-quality solutions when compared with Rossi's algorithm. In detail, FastColor obtains better solutions on 59 instances. For the remaining 32 instances, the two algorithms obtain the same solutions. Further observations show that both algorithms prove the optimal solution for these 32 instances except one, which indicates that these 32 instances might be relatively easy. In addition, FastColor proves the optimality for 64 instances, while Rossi's algorithm does so for only 31 instances.

Table 1: Results on Network Data Repository Benchmark (I)

Instance	$\|V\|$	$\|E$	FastColor			Rossi's	
			Min(Avg)	Time	#Iter.	Min	Time
bio-celegans	453	2025	**9***	< 0.01	1	10	-
bio-diseasome	516	1188	**11***	< 0.01	1	11	-
bio-dmela	7393	25569	**7***	0.01	1	8	-
bio-yeast	1458	1948	**6***	< 0.01	1	6	-
ca-AstroPh	17903	196972	**57***	0.04	1	57	-
ca-citeseer	227320	814134	87*	0.15	1	87*	-
ca-CondMat	21363	91286	26*	0.01	1	26*	-
ca-CSphd	1882	1740	3*	< 0.01	1	3*	-
ca-dblp-2010	226413	716460	75*	0.15	1	75*	-
ca-dblp-2012	317080	1049866	114*	0.32	1	114*	-
ca-Erdos992	6100	7515	8*	< 0.01	1	8*	-
ca-GrQc	4158	13422	44*	< 0.01	1	44*	-
ca-HepPh	11204	117619	239*	0.11	1	239*	-
ca-hollywood-2009	1069126	56306653	2209*	31.2	1	2209*	-
ca-MathSciNet	332689	820644	25*	0.24	1	25*	-
ca-netscience	379	914	9*	< 0.01	1	9*	-
socfb-A-anon	3097165	23667394	**26(27.3)**	35.42	14	33	-
socfb-B-anon	2937612	20959854	**24(24.8)***	24.74	10	28	-
socfb-Berkeley13	22900	852419	**42(42.5)***	9.08	33	48	0.5
socfb-CMU	6621	249959	**45***	0.41	19	47	-
socfb-Duke14	9885	506437	**40(40.2)**	14.47	41	45	-
socfb-Indiana	29732	1305757	**48(48.3)***	25.36	54	52	0.4
socfb-MIT	6402	251230	**37(37.6)**	12.07	33	43	-
socfb-OR	63392	816886	**31(31.2)**	15.24	109	34	0.9
socfb-Penn94	41536	1362220	**44***	2.93	37	47	0.5
socfb-Stanford3	11586	568309	**51***	2.89	52	55	-
socfb-Texas84	36364	1590651	**52(52.8)**	20.58	79	56	-
socfb-UCLA	20453	747604	**51***	0.98	1	53	-
socfb-UConn	17206	604867	**50***	0.96	1	51	-
socfb-UCSB37	14917	482215	**53***	0.84	1	55	-
socfb-UF	35111	1465654	**56(56.8)**	0.19	1	60	0.7
socfb-UIllinois	30795	1264421	**57***	4.48	1	58	-
socfb-Wisconsin87	23831	835946	**38(38.8)**	4.01	11	43	-
inf-power	4941	6594	6*	< 0.01	1	6*	-
inf-roadNet-CA	1957027	2760388	**4***	0.53	1	5	-
inf-roadNet-PA	1087562	1541514	**4***	0.4	1	4	-
ia-email-EU	32430	54397	**13**	0.91	15	17	-
ia-email-univ	1133	5451	12*	< 0.01	1	12*	-
ia-enron-large	33696	180811	**23(23.8)**	3.24	37	26	-
ia-enron-only	143	623	8*	< 0.01	1	8*	-
ia-fb-messages	1266	6451	**6(6.1)**	17.33	6	8	-
ia-infect-dublin	410	2765	16*	< 0.01	1	16*	-
ia-infect-hyper	113	2196	**16**	0.01	2	19	-
ia-reality	6809	7680	**5***	< 0.01	1	5	-
ia-wiki-Talk	92117	360767	**25**	17.91	30	28	-
rec-amazon	91813	125704	5*	0.02	1	5*	-

6.3 Results on SNAP and DIMCAS10

The benchmarks used in [Verma *et al.*, 2015] were originally from Stanford Large Network Dataset Collection,[3] and the 10th DIMACS implementation challenge.[4] In this subsection, we compare FastColor with Verma's algorithm on these benchmarks, which contains totally 53 instances.

The results are presented in Table 3. We focus on comparing the solution quality. For the 53 instances, FastColor performs better than Verma's algorithm on 19 instances, while worse on only 2 instances. For the remaining

[2] http://www.graphrepository.com/networks.php

[3] http://snap.stanford.edu/data

[4] http://www.cc.gatech.edu/dimacs10/

Table 2: Results on Network Data Repository Benchmark (II)

Instance	$\|V\|$	$\|E\|$	FastColor			Rossi's	
			Min(Avg)	Time	#Iter.	Min	Time
rt-retweet-crawl	1112702	2278852	13*	0.9	1	13*	-
rt-retweet	96	117	4*	< 0.01	1	4*	-
rt-twitter-copen	761	1029	**4***	< 0.01	1	5	-
soc-BlogCatalog	88784	2093195	**76(76.3)**	30.01	24	84	1.1
soc-brightkite	56739	212945	**37***	0.08	1	39	-
soc-buzznet	101163	2763066	**50**	16.85	18	59	2.2
soc-delicious	536108	1365961	**21***	0.61	1	22	5.5
soc-digg	770799	5907132	**55(55.7)**	34.47	54	63	7.5
soc-dolphins	62	159	5*	< 0.01	1	5*	-
soc-douban	154908	327162	**11***	0.14	6	13	1.4
soc-epinions	26588	100120	**17**	0.09	46	20	-
soc-flickr	513969	3190452	**94(94.6)**	35.01	108	100	4.5
soc-flixster	2523386	7918801	**35**	18.31	73	46	24.3
soc-FourSquare	639014	3214986	**31**	5.44	3	34	10.3
soc-gowalla	196591	950327	29*	0.43	1	29*	-
soc-karate	34	78	5*	< 0.01	1	5*	-
soc-lastfm	1191805	4519330	**19(19.6)**	19.25	55	25	11.4
soc-livejournal	4033137	27933062	214*	6.18	1	214*	-
soc-LiveMocha	104103	2193083	**25(25.9)**	10.18	19	30	1.6
soc-orkut	2997166	106349209	**71**	18.34	1	83	-
soc-pokec	1632803	22301964	**29***	18.14	1	31	35.2
soc-slashdot	70068	358647	**29**	0.39	36	31	0.5
soc-twitter-follows	404719	713319	**6***	0.64	6	7	2.9
soc-wiki-Vote	889	2914	**7***	< 0.01	1	8	-
soc-youtube	495957	1936748	**22(22.9)**	6.57	44	29	3.6
soc-youtube-snap	1134890	2987624	**23**	42.4	122	31	-
tech-as-caida2007	26475	53381	**16***	0.02	1	18	2.8
tech-as-skitter	1694616	11094209	**67***	9.62	104	70	25.9
tech-internet-as	40164	85123	**16***	0.03	11	18	-
tech-p2p-gnutella	62561	147878	**5**	0.07	4	7	0.5
tech-RL-caida	190914	607610	**17***	0.44	74	19	1.9
tech-routers-rf	2113	6632	**16***	< 0.01	1	17	-
tech-WHOIS	7476	56943	**58***	0.15	1	60	-
web-arabic-2005	163598	1747269	102*	**0.1**	1	102*	1.9
web-BerkStan	12305	19500	29*	< 0.01	1	29*	-
web-edu	3031	6474	30*	< 0.01	1	30*	-
web-google	1299	2773	18*	< 0.01	1	18*	-
web-indochina-2004	11358	47606	50*	< 0.01	1	50*	-
web-it-2004	509338	7178413	**432***	0.33	1	432	5.6
web-polblogs	643	2280	10	< 0.01	1	10	-
web-sk-2005	121422	334419	82*	**0.02**	1	82*	0.8
web-spam	4767	37375	**20***	0.01	1	22	-
web-uk-2005	129632	11744049	500*	**0.35**	1	500*	3.8
web-webbase-2001	16062	25593	33*	< 0.01	1	33*	-
web-wikipedia2009	1864433	4507315	31*	**1.31**	1	31*	18

Table 3: Results on SNAP and DIMCAS10 Benchmarks

Instance	$\|V\|$	$\|E\|$	FastColor			Verma's	
			Min(Avg)	Time	#Iter.	Min	Time
Amazon0302	262111	899792	7*	**0.17**	1	7*	1.3
Amazon0312	400727	2349869	11*	**0.38**	1	11*	86.73
Amazon0505	410236	2439437	11*	**0.38**	1	11*	44.64
Amazon0601	403394	2443408	11*	**0.39**	1	11*	68.32
Cit-HepPh	34546	420877	**19***	0.23	1	21	42.89
Cit-HepTh	27770	352285	**24**	0.03	1	25	29.9
cit-Patents	3774768	16518947	**11***	28.03	10	12	761.86
Email-EuAll	265214	364481	**18(18.7)**	4.38	19	20	642.18
p2p-Gnutella04	10876	39994	**5**	0.05	5	6	3.68
p2p-Gnutella24	26518	65369	**5**	0.02	4	6	19.29
p2p-Gnutella25	22687	54705	**5**	< 0.01	1	6	14.27
p2p-Gnutella30	36682	88328	**5**	0.03	4	6	33.33
p2p-Gnutella31	62586	147892	**5**	0.07	3	6	5.26
Slashdot0811	77360	469180	29	2.22	33	**29***	2.7
Slashdot0902	82168	504230	30	1.18	30	**29***	4
soc-Epinions1	75879	405740	**28(28.7)**	10.48	46	30	608.15
web-BerkStan	685230	6649470	201*	**4.26**	1	201*	27.32
web-Google	875713	4322051	44*	**1.27**	1	44*	4.47
web-NotreDame	325729	1090108	155*	**0.14**	1	155*	0.8
web-Stanford	281903	1992636	61*	**1.33**	1	61*	13.62
WikiTalk	2394385	4659565	**49**	11.89	21	51	1221.04
Wiki-Vote	7115	100762	**22**	4.65	16	24	604.67
as-22july06	22963	48436	17*	**0.01**	11	17*	0.14
caidaRouterLevel	192244	609066	17*	**0.34**	47	17*	1.44
citationCiteseer	268495	1156647	13*	**0.8**	1	13*	1.55
cnr-2000	325557	2738969	84*	**2.06**	1	84*	9.89
coAuthorsCiteseer	227320	814134	87*	**0.15**	1	87*	0.3
coAuthorsDBLP	299067	977676	115*	**0.22**	1	115*	0.36
cond-mat-2005	40421	175691	30*	**0.03**	1	30*	0.06
coPapersCiteseer	434102	16036720	845*	**1.72**	1	845*	5.17
coPapersDBLP	540486	15245729	337*	**1**	1	337*	3.79
eu-2005	862664	16138468	387*	**9.21**	1	387*	68.64
in-2004	1382908	13591473	489*	**3.74**	1	489*	22.51
kron_g500-simple-logn16	65536	2456071	**153(153.9)**	22.29	8	155	2842.85
rgg_n_2_17_s0	131072	728753	15*	0.22	1	15*	**0.17**
rgg_n_2_19_s0	524288	3269766	18*	**1.02**	1	18*	1.43
rgg_n_2_20_s0	1048576	6891620	17*	**1.45**	1	17*	1.81
rgg_n_2_21_s0	2097152	14487995	19*	5.29	1	19*	**3.7**
rgg_n_2_22_s0	4194304	30359198	20*	12.58	1	20*	**7.38**
rgg_n_2_23_s0	8388608	63501393	21*	36.48	1	21*	**14.99**
rgg_n_2_24_s0	16777216	132557200	21*	**44.73**	1	21*	50.21
uk-2002	18520486	261787258	944*	**48.83**	1	944*	330.59
333SP	3712815	11108633	5	**2**	1	5	11734.5
audikw1	943695	38354076	**42**	21.12	13	44	4391.04
belgium.osm	1441295	1549970	3*	**0.2**	1	3*	6.33
cage15	5154859	47022346	**12(12.4)**	36.45	4	13	36141.6
ecology1	1000000	1998000	2*	**0.28**	0	2*	2.65
G_n_pin_pout	100000	501198	**6**	0.28	2	8	22.17
ldoor	952203	22785136	**33**	4.52	0	35	4474.09
luxembourg.osm	114599	119666	3*	**0.01**	1	3*	12.99
preferential-Attachment	100000	499985	6*	0.48	0	6*	**0.39**
smallworld	100000	499998	**7(7.9)**	2.74	1	8	35.81
wave	156317	1059331	**8**	0.26	2	9	98.59

32 instances, FastColor and Verma's algorithm obtain the same solution quality, and prove the optimal solution for all except one instance (333SP), indicating these 32 instances might be relatively easy.

It is not so scientific to compare the runtime under different platform. Nevertheless, the significant gap still demonstrates the superiority of FastColor in terms of run time. FastColor is much faster than Verma's algorithm. With regard to averaged time, FastColor is 10 times faster for 27 instances and 100 times faster for 18 instances. In particular, Verma's algorithm needs more than 10 thousand seconds to obtain a 5 colors assignment for 333P, while FastColor only need 2 seconds.

7 Summary and Future Work

This paper presented a novel graph coloring algorithm for coloring massive graphs within short time. We proposed a reduction rule which is based on a novel concept called degree bounded independent set. The method iteratively executes this rule by interleaving between lower bound computation and graph reduction. Experiments on real-world large graphs show that FastColor is very fast and finds better solutions than state of the art algorithms.

We would like to explore more reduction rules for GCP, such as heuristic rules. Another direction is to apply similar method to other graph problems.

Acknowledgements

This work is partially supported by the National Natural Science Foundation of China under Grant 61502464, Grant 61472369, and Grant 61370072, partially supported by the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing under Grant 2016A06, and partially supported by the Australian Research Council under Grant DP150101618. Shaowei Cai is also supported by Youth Innovation Promotion Association, Chinese Academy of Sciences.

References

[Batagelj and Zaversnik, 2003] Vladimir Batagelj and Matjaz Zaversnik. An o(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.

[Brélaz, 1979] Daniel Brélaz. New methods to color the vertices of a graph. *Commun. ACM*, 22(4):251–256, April 1979.

[Cai and Lin, 2016] Shaowei Cai and Jinkun Lin. Fast solving maximum weight clique problem in massive graphs. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 568–574, 2016.

[Cai, 2015] Shaowei Cai. 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, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 747–753, 2015.

[Campêlo *et al.*, 2008] Manoel B. Campêlo, Victor A. Campos, and Ricardo C. Corrêa. On the asymmetric representatives formulation for the vertex coloring problem. *Discrete Applied Mathematics*, 156(7):1097–1111, 2008.

[Chow and Hennessy, 1990] Fred C. Chow and John L. Hennessy. The priority-based coloring approach to register allocation. *ACM Trans. Program. Lang. Syst.*, 12(4):501–536, 1990.

[de Werra, 1985] Dominique de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19:151–162, 1985.

[Galinier *et al.*, 2013] Philippe Galinier, Jean-Philippe Hamiez, Jin-Kao Hao, and Daniel Cosmin Porumbel. Recent advances in graph vertex coloring. In *Handbook of Optimization - From Classical to Modern Approach*, pages 505–528. Springer, 2013.

[Garey and Johnson, 1979] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, San Francisco, CA, USA, 1979.

[Gualandi and Malucelli, 2012] Stefano Gualandi and Federico Malucelli. Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing*, 24(1):81–100, 2012.

[Hansen *et al.*, 2009] Pierre Hansen, Martine Labbé, and David Schindl. Set covering and packing formulations of graph coloring: Algorithms and first polyhedral results. *Discrete Optimization*, 6(2):135–147, 2009.

[Hao and Wu, 2012] Jin-Kao Hao and Qinghua Wu. Improving the extraction and expansion method for large graph coloring. *Discrete Applied Mathematics*, 160(16-17):2397–2407, 2012.

[Kearns *et al.*, 2006] Michael Kearns, Siddharth Suri, and Nick Montfort. An experimental study of the coloring problem on human subject networks. *Science*, 313(5788):824–827, 2006.

[Leighton, 1979] F.T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–503, 1979.

[Malaguti *et al.*, 2011] Enrico Malaguti, Michele Monaci, and Paolo Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, 8(2):174–190, 2011.

[Newman, 2003] Mark E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.

[Peng *et al.*, 2016] Yun Peng, Byron Choi, Bingsheng He, Shuigeng Zhou, Ruzhi Xu, and Xiaohui Yu. Vcolor: A practical vertex-cut based approach for coloring large graphs. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, pages 97–108, 2016.

[Rossi and Ahmed, 2014] Ryan A. Rossi and Nesreen K. Ahmed. Coloring large complex networks. *Social Netw. Analys. Mining*, 4(1):228, 2014. Social Netw. Analys. Mining.

[Rossi and Ahmed, 2015] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[Seidman, 1983] S. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983.

[Shen *et al.*, 2012] Yilin Shen, Dung T. Nguyen, Ying Xuan, and My T. Thai. New techniques for approximating optimal substructure problems in power-law graphs. *Theor. Comput. Sci.*, 447:107–119, 2012.

[Tomita *et al.*, 2010] Etsuji Tomita, Yoichi Sutani, Takanori Higashi, Shinya Takahashi, and Mitsuo Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique. In *WALCOM: Algorithms and Computation, 4th International Workshop, WALCOM 2010, Dhaka, Bangladesh, February 10-12, 2010. Proceedings*, pages 191–203, 2010.

[Verma *et al.*, 2015] Anurag Verma, Austin Buchanan, and Sergiy Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks. *INFORMS Journal on Computing*, 27(1):164–177, 2015.

[Wu and Hao, 2012] Qinghua Wu and Jin-Kao Hao. Coloring large graphs based on independent set extraction. *Computers & OR*, 39(2):283–290, 2012.

[Zuckerman, 2007] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007.