Computers & Operations Research 51 (2014) 282–301

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

An exact algorithm with learning for the graph coloring problem ☆

Zhaoyang Zhou [a,c], Chu-Min Li [a,b], Chong Huang [a], Ruchu Xu [a,*]

[a] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China
[b] MIS, Université de Picardie Jules Verne, 33 Rue St. Leu 80039 Amiens Cedex, France
[c] Library, Hubei University, Wuhan, China

ARTICLE INFO

Available online 24 May 2014

Keywords:
Backtracking
Clause learning
Graph Coloring
SAT

ABSTRACT

Given an undirected graph $G=(V,E)$, the Graph Coloring Problem (GCP) consists in assigning a color to each vertex of the graph G in such a way that any two adjacent vertices are assigned different colors, and the number of different colors used is minimized. State-of-the-art algorithms generally deal with the explicit constraints in GCP: any two adjacent vertices should be assigned different colors, but do not specially deal with the implicit constraints between non-adjacent vertices implied by the explicit constraints. In this paper, we propose an exact algorithm with learning for GCP which exploits the implicit constraints using propositional logic. Our algorithm is compared with several exact algorithms among the best in the literature. The experimental results show that our algorithm outperforms other algorithms on many instances. Specifically, our algorithm allows to close the open DIMACS instance 4-Fullins_5.

1. Introduction

Given an undirected graph $G=(V, E)$, where V is a set of vertices and E a set of unordered pairs of vertices called edges $\{(v_i, v_j)|v_i \in V, v_j \in V\}$, the Graph Coloring Problem (GCP) consists in assigning a color to each vertex of the graph G in such a way that any two adjacent vertices are assigned different colors, and the number of different colors used is minimized. The minimum number of colors needed to color G is called the *chromatic number* of G, denoted by $\chi(G)$. Deciding whether a graph can be colored using at most k colors is referred as *k-colorability* problem, which is the decision version of GCP.

GCP is one of the most studied NP-hard combinatorial optimization problems. It is interesting not only for studying the computational complexity but also for practical applications. Many real world problems can be naturally encoded into GCP and solved using a GCP algorithm. Examples of such problems include scheduling [1,2], timetable [3,4], register allocation [5], frequency assignment [6], and communication networks [7].

Algorithms for GCP generally fall into two categories: exact algorithms (see e.g., [8–12]) and approximate algorithms. Approximate algorithms include construction method [13], local search [14], evolutionary algorithms based on populations [15,16], hybrid algorithms [17], and other heuristic methods [18]. Because of its hardness, most algorithms for GCP in the literature are approximate and few are exact. Indeed, a GCP encoding a real application problem is often very large and can be out of the reach of exact algorithms, because exact coloring algorithms generally need to enumerate the search space of a problem and have difficulties to solve large instances. However, approximate algorithms are usually unable to prove the optimality of their solutions. Especially, they are usually unable to prove the unfeasibility of a k-colorability problem. So exact algorithms remain very useful and indispensable. In this paper, we focus on exact algorithms for GCP.

The constraints in a GCP are very easy to express and understand, saying that any two adjacent vertices cannot be assigned the same color. However, these simple explicit constraints between adjacent vertices imply many complex constraints among non-adjacent vertices. It appears that most state-of-the-art GCP algorithms do not specially detect and exploit these implicit constraints. One way to discover and exploit the implicit constraints is to encode a GCP using k colors into the Boolean satisfiability problem (SAT) by expressing the explicit constraints in CNF (Conjunctive Normal Form) clauses. Then modern SAT algorithms discover and exploit the implicit constraints among non-adjacent vertices using Conflict Driven Clause Learning (CDCL). However, the SAT instance encoding the GCP generally is very large, limiting the size of the GCP that can be solved in this way.

☆This work is supported by National Natural Science Foundation of China (NSFC). (Grant No. 61070235)

* Corresponding author.
E-mail addresses: authar@163.com (Z. Zhou), chu-min.li@u-picardie.fr (C.-M. Li), jason_hch@hotmail.com (C. Huang), xrcy0315@sina.com (R. Xu).

http://dx.doi.org/10.1016/j.cor.2014.05.017
0305-0548/© 2014 Published by Elsevier Ltd.

In this paper, we propose a new exact algorithm for GCP based on a backtracking schema. The explicit constraints of a GCP are not encoded into SAT. At the beginning, every vertex has the set of k available colors. Vertices are assigned a color from their sets of available colors one by one. When a color c is assigned to a vertex v, c is removed from the set of available colors of all vertices adjacent to v. Each time a dead-end is encountered, i.e., when the set of available colors of a vertex becomes empty, the Conflict Driven Clause Learning (CDCL) schema in modern SAT solvers is adapted to discover an implicit constraint among vertices. This implicit constraint is used in subsequent search to reduce the search space.

The paper is organized as follows. Section 2 presents the related work. Section 3 presents the motivation of our approach. Section 4 presents the new backtracking algorithm and discusses in detail the clause learning schema in our algorithm adapted from modern SAT solvers. Section 5 empirically studies the clause management strategies and the distribution and impact of the learnt clauses in the real coloring process, and shows the effectiveness of CDCL by comparing our algorithms (with and without clause learning) with other algorithms on standard GCP benchmarks. Section 6 concludes the paper.

2. Related work

GCP can be expressed using integer programming and then solved using a software such as CPLEX. For each vertex v and each color c, let v_c be a binary variable that evaluates to 1 if and only if (iff) v is assigned color c. The decision version of GCP, i.e., the k-colorability problem, can be naturally expressed using $n*k$ binary variables as follows:

$$\sum_c v_c = 1, \quad \forall\ v \in V \tag{1}$$

$$u_c + v_c \le 1, \quad \forall\ (u,v) \in E \text{ and } \forall\ c \tag{2}$$

Eq. (1) says that each vertex is assigned exactly one color, and Eq. (2) says that two adjacent vertices cannot be assigned the same color. In order to minimize the number of colors, one can consider n colors and use n additional binary variables y_c $(1 \le c \le n)$ that evaluate to 1 iff color c is used. The integer programming problem becomes

$$\min \sum_c y_c$$

subject to Eq. (1) and

$$u_c + v_c \le y_c, \quad \forall\ (u,v) \in E \text{ and } \forall\ c \tag{3}$$

The number of binary variables in the above integer programming formulations of GCP is in $O(n^2)$. Mehrotra and Trick proposed a formulation using up to an exponential number of variables in [19]. Let S be the set of all independent sets of G, a binary variable x_s is associated to each independent set s that evaluates to 1 iff all vertices in s are assigned the same color. The corresponding integer programming formulation of GCP becomes

$$\min \sum_{s \in S} x_s$$

subject to

$$\sum_{v \in s, s \in S} x_s \ge 1, \quad \forall\ v \in V \tag{4}$$

An alternative solving approach of GCP is to enumerate all possible vertex colorings of G following a branch-and-bound schema. Let G be a graph of n vertices such that each vertex v is associated with a set $C_v = \{1, 2, \ldots, n\}$ of n available colors, $G \backslash v$ be G after removing the vertex v and all incident edges of v, and let function max(a, b) denote the largest value between a and b. Algorithm 1 returns the chromatic number $\chi(G)$ after enumerating all possible colorings of G in a backtracking tree.

Algorithm 1. backGCP(G, k, UB), a backtracking algorithm for GCP.

```
Input: A graph $G=(V, E)$, the largest color $k$ used so far in the current partial coloring solution, and the smallest number $UB$ of colors
    used in a complete coloring solution of $G$ so far
Output: the chromatic number $\chi(G)$ of $G$
begin
  if $|V| = 0$ then
    return $k$;
  if there is a vertex $v$ such that $C_v$ does not contain any color less than $UB$ then
    return $UB$;
  select a vertex $v$ from $G$; /* branching */
  for each color $c$ in $C_v$ such that $c < UB$ do
    remove $c$ from $C_u$ of each vertex $u$ adjacent to $v$;
    $UB = \text{backGCP}(G \backslash v, \max(k, c), UB)$;
    insert $c$ to$C_u$ of each vertex $u$ adjacent to $v$;
  return $UB$
end
```

Algorithm 1 should be called with backGCP(G, 0, $n+1$) to search for the chromatic number of G. At the beginning, $k=0$, i.e., the largest color used is 0 (no color is used at the beginning), the smallest number UB of colors used in a complete coloring solution so far is initialized to $n+1$, and all the vertices have the same set of n available colors. The algorithm selects a vertex v, and for each available color c of v smaller than UB, assigns c to v, i.e., removes c from the adjacent vertices of v, and recursively calls the algorithm to color the remaining vertices of G. If G does not contain any