International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 10 (2015) pp. 25939-25944

http://www.ripublication.com

Solution to Graph Coloring Problem Using Heuristics and Recursive Backtracking

Raja Marappan,
Department of Computer Applications,
SASTRA University,
Thanjavur - 613401, India
e-mail: raja_csmath@cse.sastra.edu
Gopalakrishnan Sethumadhavan,
Department of Computer Applications,
SASTRA University,
Thanjavur - 613401, India
e-mail: sgk@mca.sastra.edu

Abstract

Graph coloring is a classical NP-Complete combinatorial optimization problem and it is widely applied in different engineering applications. This paper explores the effectiveness of applying heuristics and recursive backtracking strategy to solve the coloring assignment of a graph G. The proposed method applies heuristics through recursive backtracking to obtain the approximate solution to $\chi(G)$, the minimum number of colors needed to color the vertices of G. The proposed heuristics splits V(G) into higher degree and lower degree vertices such that the search space is reduced when calling the recursive backtracking algorithm for the higher degree vertices first. The performance of this approximation method is evaluated using some well known benchmark graphs and the results are presented.

Keywords: Graph coloring, heuristics, backtracking, NP-Complete

Introduction

An undirected simple graph G is defined as G: (V, E), which consists of the vertex set V(G): $\{v_1, v_2, \ldots, v_n\}$ and an edge set E(G): $\{e_1, e_2, \ldots, e_m\}$ such that every edge is incident with the unique end vertices (v_i, v_j). The adjacency matrix of G is defined as A(G) which is an $n \times n$ symmetric binary matrix where $A_{ij} = 1$ if there is an edge between the vertices v_i and v_j and also called the adjacent vertices; and $A_{ij} = 0$, otherwise.

The smallest number of colors, $\chi(G)$ required to color V(G) with no two adjacent vertices assigned with the same color is a Graph Coloring Problem (GCP) [2]. When G is a simple graph then obtaining $\chi(G)$ takes the solution space of n! using the approximation methods. GCP is used in different applications such as register allocation, scheduling, channel assignment and noise reduction [3] etc. GCP is a NP-hard problem, hence there is no method devised to solve it in a polynomial time [1]. Hence a fast and effective approximation algorithm is required to find $\chi(G)$ because of its NP-completeness. Approximation to GCP is found using some of the existing methods such as genetic algorithms [4], [5], [6], [10], [11], [12], [13], [14], branch-and-bound, branch-and-cut, and backtracking algorithms [7], [8], [9].

This paper presents new heuristics based recursive backtracking method to find the value of $\chi(G)$ and $\chi(G)$ coloring assignment of a given G. Initially the backtracking procedure splits the vertices of G into two sets: higher degree vertices and lower degree vertices. The higher degree vertices are initially colored and then lower degree vertices are colored using the proposed recursive backtracking procedure. This procedure has been tested with some benchmark graphs such as queen, Mycielski, huck.col, jean.col, games120.col, miles250.col, david.col, anna.col and the results are obtained to be effective.

Heuristics and Recursive Backtracking

The colors of a given graph G is denoted in the integer set {1, 2, ..., c} and the solutions are given by the color[i] such that $1 \leq i \leq n$. The recursive backtracking algorithm CCOLORING is developed using recursion and heuristic strategies and presented in Algorithm 1 and Algorithm 2. The state space tree is also constructed with degree (c+1) with the height of (n+1). Clearly all the level i nodes have c offsprings corresponding to the c feasible assignments to color[i], $1 \leq i \leq n$. The leaf or terminal nodes are at level n+1. Figure 2 depicts the state space coloring search tree when n = c = 3.

Algorithm 1: Finding all c colorings of G

```
Algorithm CCOLORING (k)
// Recursive schema of backtracking
// V(G) assigns 1, 2, ... , c such that adjacent vertices are assigned distinct colors
// the subscript of the next vertex to assign color is k
// c, n, color[i] are global integers
{
  Repeat the following operation:
     Produce all valid values of color[k];
     Call NEXTCOLVALUE(k);  // assign a right color to color[k]
     If (color[k] = 0) then exit;       // new color is impossible
     If (k-n = 0) then
             Print(color[i]);           //  maximum c integers are assigned to V(G)
     Else
             Call CCOLORING(k + 1);
Loop repeat;
```

}

Algorithm CCOLORING (k) is begun by setting the color[i], 1≤ i ≤ n is initialized to 0. Then the statement call CCOLORING (1) is invoked.

Algorithm NEXTCOLVALUE produces the possible colors for color[k], after defining color[1] through color[*k* – 1]. The control logic of CCOLORING picks an element from the set of possibilities repeatedly and assigns it to color[k] with the recursive call of the designed CCOLORING algorithm.

Algorithm 2: Generating a Next Color

```
Algorithm NEXTCOLVALUE (k)
// color[1], color[2], …, color[k-1] is assigned integers in [l, c] with nearby
// vertices have dissimilar integers. A value for color[k] is determined in the range
// [0, c].
// assigning color[k] to the next maximum color. Otherwise color[k] = 0.
{
 Repeat the following operation:
    color[k] = (color[k]+1) % (c + 1)      // next maximum color
    If (color[k] = 0) then return;          // all integers are exhausted
    For j=1 to n do          // check if this color is distinct from adjacent colors
            If (A[k, j] = 1) then
            If (color[j] = color[k]) then exit;
    If (j-n = 1) then return;
Loop Repeat;
}
```

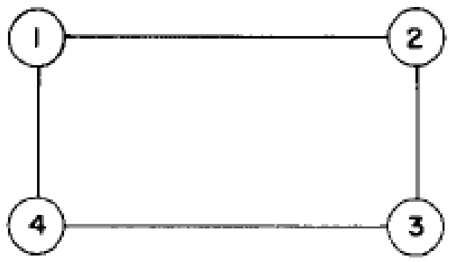

Figure 1: A simple graph - G with n=4

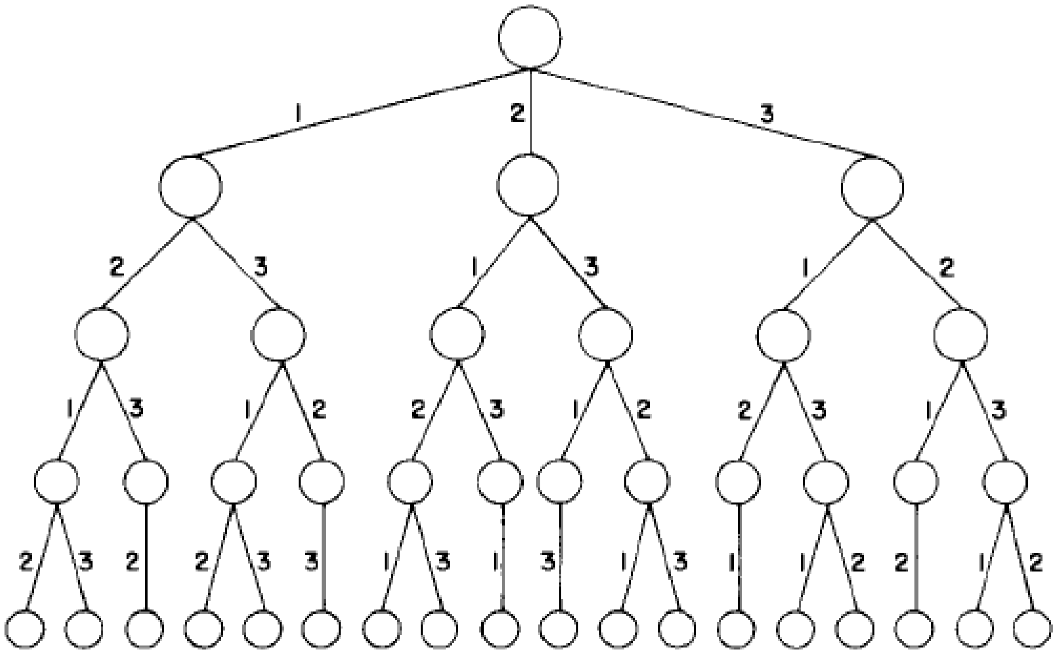

Figure 2: State space search tree – all possible C COLORINGS of G where c=3

Figure 1 shows a simple graph with n = 4 vertices. The tree which is generated by the algorithm CCOLORING is shown in Figure 2. Every path to a leaf represents a coloring using at most c = 3 colors. There are 12 solutions exist with exactly c = 3 colors.

Computational Complexity Analysis

The computational complexity of the proposed graph coloring algorithm is analyzed and presented.

Theorem 1: The computational complexity of CCOLORING is $\Theta(n.c^n)$.

Proof: The computing time of the proposed coloring method is analyzed by computing the number of internal nodes present in state space search tree. Clearly the total number of nodes in that tree is

$$\sum_{j=0}^{n-1} c^j = c^0 + c^1 + c^2 + \ldots c^{n-1}$$

Every internal node requires O(cn) computational complexity in the NEXTCOLVALUE method to evaluate the offsprings to assign valid assignments of integers. Thus the overall computing time is given by

$$\sum_{j=1}^{n} n.c^j = n\,(c^1 + c^2 + \ldots c^n) = (c^{n+1}-1)/(c-1) = \Theta(n.c^n)$$

Hence the computational complexity of the proposed method is $\Theta(n.c^n)$.

Simulation and Results on Some Benchmark Graphs

The proposed method is simulated on some of the benchmark graphs using Intel Core i5-2450M 2.5GHz with Turbo boost up to 3.1GHz system in Java JDK 1.7 environment. Algorithm has been implemented using JAVA language and the results are tabulated.

Table I COMPUTATION OF χ(G) ON BENCHMARK GRAPHS

Graph No	Graph (G)			Minimum color obtained in the proposed method
	Graph Type	Instances	χ(G)	
1	queen5_5.col	n=25; m=320	5	5
2	queen6_6.col	n=36; m=580	7	7
3	queen7_7.col	n=49; m=952	7	7
4	queen8_8.col	n=64; m=1456	9	9
5	myciel5.col	n=47; m=236	6	6
6	myciel6.col	n=95; m=755	7	7
7	huck.col	n=74; m=301	11	11
8	jean.col	n=80; m=254	10	10
9	david.col	n=87; m=406	11	11
10	games120.col	n=120; m=638	9	9

Table I shows the computation of χ(G) on some of the benchmark graphs. The following conclusions are drawn from this simulation:

1. The proposed heuristics and recursive backtracking obtains the exact solution.
2. V(G) is split into higher degree and lower degree vertices such that the coloring task is simplified when calling the recursive backtracking algorithms for higher degree vertices first.
3. When n increases, χ(G) also increases for some benchmark graphs.
4. Graph instances affects the computational complexity of the recursive backtracking.
5. Space complexity is increased while computational time is reduced.

Conclusion

The new heuristics using recursive backtracking technique to solve graph coloring is presented in this paper. The heuristics splits V(G) into higher degree and lower degree vertices such that the search space is reduced when the recursive backtracking algorithm is initially applied for higher degree vertices. The simulation is conducted on several benchmark graphs to evaluate its computational complexity. The simulated results of this proposed algorithm are presented. The proposed heuristics and recursive backtracking obtains the exact solution for most of the benchmark graphs. When n increases, χ(G) also increases for some benchmark graphs, which results in a near optimal performance of this coloring algorithm. Graph instances affects the computational complexity of the recursive backtracking. Space complexity is increased while computational time is reduced by applying the proposed heuristics.

References

[1] Garey, M. R. and Johnson, D. S., Computers and Intractability: Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

[2] Tommy R. Jensen, Bjarne Toft, Graph Coloring Problems, Wiley-Inter science, 1995.

[3] Noise Reduction in VLSI Circuits using Modified GA Based Graph Coloring - International Journal of Control and Automation, Vol. 3, No. 2, June, 2010.

[4] A. Hertz, and D. E Werra. "Using tabu search techniques for graph coloring", Computing Vol. 39, No. 4, pp. 345-351, 1987.

[5] C. Fleurent and J. A. Ferland. "Genetic and hybrid algorithms for graph coloring", Annals of Operations Research, 63, pp. 437-463, 1995.

[6] C. L.Mumford. "New order-based crossover for the graph coloring problem", T. P. Runarsson et al. (Eds.): PPSN LX, vol. 4193, pp. 80-88, 2006.

[7] Anuj Mehrotra and Michael A. Trick. A column generation approach for graph coloring. INFORMS Journal on Computing, 8(4):344–354, 1996.

[8] Isabel Méndez-Diaz and Paula Zabala. A branch-and-cut algorithm for graph coloring. Discrete Applied Mathematics, 154(5):826–847, 2006.

[9] R. Monasson. On the analysis of backtrack procedures for the coloring of random graphs. In E. Ben-Naim, H. Frauenfelder, and Z. Toroczkai, editors, Complex Networks, pages 235–254. Springer, 2004.

[10] Lixia Han and Zhanli Han, A Novel Bi-objective Genetic Algorithm for the Graph Coloring Problem, 2010 Second International Conference on Computer Modeling and Simulation.

[11] Tamás Szép and Zoltán Ádám Mann, Graph coloring: the more colors, the better?, CINTI 2010, 11th IEEE International Symposium on Computational Intelligence and Informatics, 18–20 November, 2010, Budapest, Hungary.

[12] Raja Marappan, Gopalakrishnan Sethumadhavan, A new genetic algorithm for graph coloring, In CIMSim2013, 5th International Conference on Computational Intelligence, Modelling and Simulation, Seoul, South Korea, pages 49-54, 2013.

[13] Gopalakrishnan Sethumadhavan, Raja Marappan, A Genetic Algorithm for Graph Coloring using Single Parent Conflict Gene Crossover and Mutation with Conflict Gene Removal Procedure, 2013 IEEE International Conference on Computational Intelligence and Computing Research, India, pages 350-355, 26-28 December 2013.

[14] Raja Marappan, Gopalakrishnan Sethumadhavan, Solution to Graph Coloring Problem using Evolutionary Optimization through Symmetry-Breaking Approach, International Journal of Applied Engineering Research, 2015, Research India Publications.