

A SIMPLE AND FAST HEURISTIC ALGORITHM FOR EDGE-COLORING OF GRAPHS*

M.A. FIOL AND J. VILALLELLA
Department de Matemàtica Aplicada 4
Universitat Politècnica de Catalunya
BarcelonaTech, Barcelona, Catalonia
e-mail: {fiol,joan.vilaltella}@ma4.upc.edu

Abstract

A simple but empirically efficient heuristic algorithm for the edge-coloring of graphs is presented. Its basic idea is the displacement of ‘conflicts’ (repeated colors in the edges incident to a vertex) along paths of adjacent vertices whose incident edges are recolored by swapping alternating colors (that is, doing a Kempe interchange). The results of performance tests on random cubic and Δ -regular graphs are presented, and a full implementation of the algorithm is given to facilitate its use and the reproducibility of results.

Keywords: cubic graph, regular graph, edge-coloring, snark, conflicting vertex,
heuristic algorithm.

2000 Mathematics Subject Classification: 05C15, 68W20.

1. Introduction

Deciding if a graph can be Δ -edge-colored (that is, whether or not it is of class one) is known to be NP -complete even for maximum degree $\Delta = 3$ (Holyer [5]). Therefore, assuming $P \neq NP$, the problem of 3-edge-coloring a cubic graph becomes intractable as the number of its vertices increases. Possible ways to ‘break’ intractability are restricting the problem or admitting probabilistic solutions. Here we take the second approach, proposing a simple and fast heuristic algorithm to 3-edge-color a cubic graph G . Experimental data suggest that, when G is 3-edge-colorable (an asymptotically almost sure property by the results of Robinson and Wormald [8]), our algorithm efficiently gives an explicit solution and, hence, allows us to break intractability. The algorithm is easily generalized to graphs with maximum degree Δ . All graphs are assumed to be connected and with no loops or multiple edges.

*Research supported by the Ministerio de Educación y Ciencia, Spain, and the European Regional Development Fund under project MTM2011-28800-C02-01 and by the Catalan Research Council under project 2005SGR00256.

2. Description of the algorithm for cubic graphs

The heuristic algorithm is called *CVD* (for ‘conflicting vertex displacement’) and starts with a random coloring of the edges of G using 3 colors. A *conflicting vertex* is a vertex whose incident edges are not properly colored (at least two of them are equally colored). The main loop of the algorithm consists of the following steps. First, randomly choose a conflicting vertex and take one of the improperly colored edges as the starting point of a Kempe chain. That is, a 2-edge-colored path whose two colors are the original improper color of the starting edge and a new color that makes the vertex unconflicting or, at least, reduces the number of edges with repeated color (a suitable new color always exists, and can be randomly chosen when it is not unique). Then, swap the two colors along the chain. Such an operation does not create new conflicts but moves the conflict along the path. Stop the color swapping when, either, another conflicting vertex is reached, where the conflicts may cancel out, or the number of swaps attains a given limit (at most the number of vertices in the graph). Next, choose again a conflicting (not necessarily different) vertex and start a new Kempe chain. Repeat the procedure until, either, all conflicts are solved (so getting a 3-edge-coloring) or a fixed limit (made precise below) is reached.

The number of conflicting vertices may stabilize even if G is not a snark (a class two graph, see Fiol [2], Gardner [4], or Isaacs [6]) because not every color swap succeeds in solving one conflict. To avoid an infinite loop, we keep a counter of the number of times a new conflicting vertex is chosen and reset this counter to zero only when the number of conflicting vertices decreases. If the counter reaches a certain constant value R , or *repetition limit*, then the main loop is restarted from a new initial random coloring of the edges. The maximum number of times the main loop can be restarted is the *iteration limit* L . If we reach this limit, we stop and assume that the cubic graph can not be properly 3-edge-colored. Then the non-null probability of making a wrong assumption is the price for ‘breaking’ intractability.

3. Complexity analysis

Assuming the worst case, every time a conflicting vertex is chosen at most n colors are swapped along a Kempe chain, and at most all n vertices in the graph could be conflicting. If the number of conflicting vertices ceases to decrease, the choice of a new conflicting vertex can be repeated no more than R times, the repetition limit, before the main loop restarts from a new random coloring of the edges, and the number of restarts is at most the iteration limit L . The cost of randomly coloring the edges of the graph is $O(n)$, and at most L random colorings must be done. The parameters R and L are fixed during the execution of the algorithm. Therefore, the expected running time is $O(n^2) + O(n) = O(n^2)$. Given the pessimistic estimation, it could be better in practice (see Section 6). A similar complexity analysis is valid in more general cases (Section 5).

4. Algorithm modifications for Δ -regular graphs and graphs with maximum degree Δ

The basic idea (conflict displacement) is the same, but this case is slightly more complex. We measure the ‘conflict level’ of a vertex (how far it is from having its incident edges properly colored) by its degree (Δ if the graph is Δ -regular) minus the number of different colors used in its incident edges. This gives a measure of the color repetition: If the number of different colors used in the incident edges of a vertex is equal to the degree of that vertex, then no color is repeated and the conflict level is 0. If the same color is used in every incident edge of vertex with degree Δ , then there are $\Delta - 1$ repetitions of that color, and the conflict level of the vertex is $\Delta - 1$. Thus, the range of possible conflict level values is $0, 1, 2, \dots, \Delta - 1$. A data structure, a dictionary to be precise, mapping each conflict level value to the set of vertices having it, is updated as the Kempe switches change the coloring of the edges (see Section 8). This information is needed because in the new algorithm conflicting vertices are chosen according to their conflict level: vertices with the highest conflict level are chosen first. The conflict level value 0 and its corresponding set of unconflicting vertices do not need to be present in the dictionary.

Instead of the number of conflicting vertices, the total number of conflicts is used to measure the ‘conflictivity’ of the graph: this is the sum of the conflict levels over all vertices in the graph, and it may be larger than the number of vertices in the graph.

To reduce the conflictivity of the initial arbitrary edge-coloring a simple greedy algorithm can be used instead of the random coloring that was sufficient in the cubic case. Besides this, and implementation details aside, the conflicting vertex choice (based on the highest conflict level) and the conflictivity measure (based on the sum of all conflict levels) are the only relevant modifications of the algorithm done to deal with the general case.

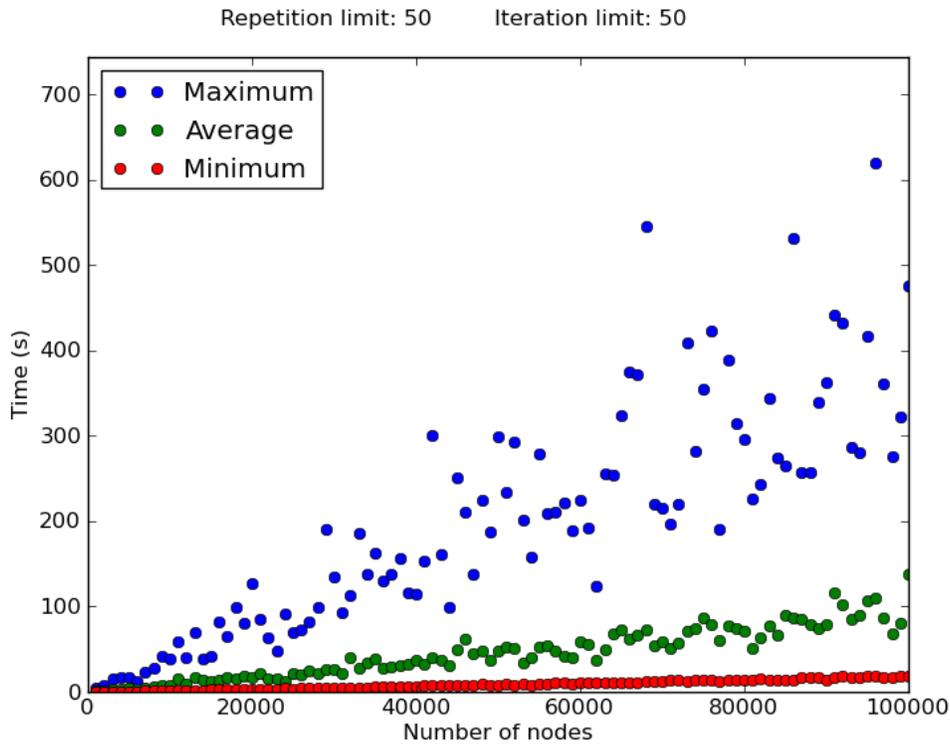
5. Complexity analysis in the case of Δ -regular graphs or graphs with maximum degree Δ

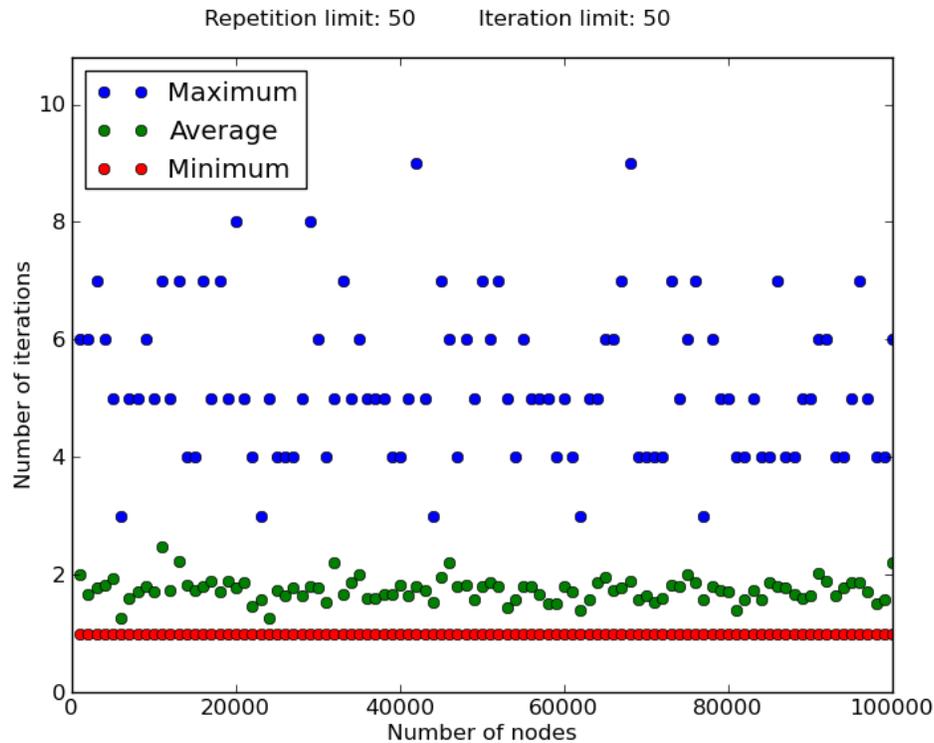
As in the previous analysis, and assuming the worst case again, every time a conflicting vertex is chosen at most n swaps are done along a Kempe chain. In the present case we use the sum of all conflict levels to measure the conflictivity of the graph, which is at most $n(\Delta - 1)$. If the conflictivity measure ceases to decrease, the choice of a new conflicting vertex can be repeated no more than R times, the repetition limit, before the main loop restarts from a new pre-coloring of the edges, and the number of restarts is at most the iteration limit L . The cost of pre-coloring all $O(\Delta n)$ edges of the graph using a random coloring, or the greedy algorithm, is $O(\Delta^2 n)$, or possibly better depending on the efficiency of sampling from the set of $O(\Delta)$ available colors, and at most L pre-colorings must be done. The parameters R and L are fixed during the execution of the algorithm. Therefore, the expected running time is $O((\Delta - 1)n^2) + O(\Delta^2 n) = O(\Delta^2 n^2)$. As in the previous case, the estimation is pessimistic and the performance could be better in practice (see Section 7).

Roughly speaking, our approach, with its vertex-centered measure of conflict, is similar to the edge-centered approach by Lee, Wan, and Guan ([7]). These authors improved on previous Δ -edge-coloring algorithms by giving a polynomial (albeit probabilistic) algorithm. The *CVD* algorithm is simpler and easy to implement (see Section 8).

6. Performance testing in the case of cubic graphs

We tested the performance of the heuristic algorithm by measuring the time needed to 3-edge-color random cubic graphs. More specifically, for each randomly generated cubic graph, we measured the time needed to find a 3-edge-coloring, the number of iterations of the main loop, and the average time per iteration (the coloring time divided by the number of iterations). For each number of vertices, we obtained the minimum, average, and maximum values of each of these magnitudes over a set of 30 random instances, and we plotted the results. It seems that the heuristic algorithm finds a 3-edge-coloring, if any, with high probability.

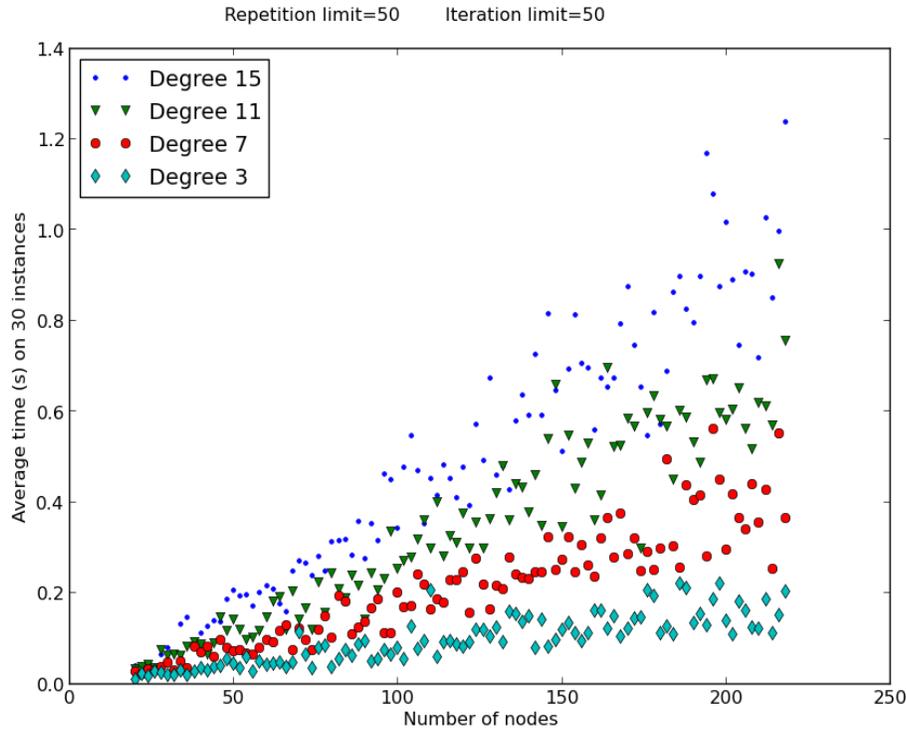




The average running time appears to grow linearly with the number of vertices (see the first figure). The same happens with the average time per iteration (not shown), whereas the average number of iterations does not seem to depend on the number of vertices (see the second figure). A linearly growing average running time is in agreement with the $O(n^2)$ expected running time.

7. Performance testing in the case of Δ -regular graphs

We followed the same testing procedure as in the cubic case (measuring the average running time over 30 instances), but varying the number n of vertices and the value of Δ . The growth of the average running time is in agreement with the expected $O(\Delta^2 n^2)$ behaviour. The following figure shows four plots, corresponding to four different values of the variable Δ (3, 7, 11 and 15).



8. Implementation details

We used the Python language (version 2.7.3) to implement the heuristic algorithm. This language allows us to use the NetworkX package to generate random regular graphs, and the matplotlib package to plot the results of the performance tests.

In order to facilitate the reproducibility of our results, or the use of the *CVD* algorithm to anyone interested, we give a complete Python implementation of the algorithm described in Section 4, depending only on the Python Standard Library and the NetworkX package. The plot of Section 6 was created using the algorithm described in Section 3, specific for cubic graphs, with a very similar (slightly simpler) implementation.

```
### Implementation data ###
```

```
#Operating system: Linux-3.2.0-31-generic-x86_64-with-Ubuntu-12.04-precise
#CPU: Intel(R) Core(TM) i3 CPU          M 370 @ 2.40GHz. Cache size: 3072 KB
#CPU speed: 933.000 Hz
```

```
### Python code ###
```

```
from random import choice,randint
import networkx as nx
```



```

def KempeNext(G,last,node,new_color,conflict_dictionary):
    available_for_next=[w for w in G[node] if w!=last
                        and G[node][w]==new_color]
    if available_for_next==[]: next_node=None
    else: next_node=choice(available_for_next)
    old_color=G[last][node]
    conflict_level_variation=colorEdgeAndUpdate(G,last,node,new_color,
                                                conflict_dictionary)
    return conflict_level_variation,old_color,next_node

def KempeStep(G,last,node,new_color,conflict_dictionary):
    conflict_level_variation,old_color,next_node=KempeNext(G,last,node,
                                                         new_color,
                                                         conflict_dictionary)
    if conflict_level_variation<0 or next_node==None: return node,None,None
    return node,next_node,old_color

def KempeProcess(G,last,node,new_color,conflict_dictionary):
    Kempe_chain=set([])
    while new_color!=None and last not in Kempe_chain:
        Kempe_chain.add(last)
        last,node,new_color=KempeStep(G,last,node,new_color,
                                     conflict_dictionary)

def KempeStart(G,D,node,conflict_dictionary):
    colors=set(range(D))
    next_node=None
    for adjacent in G[node]:
        edge_color=G[node][adjacent]
        if edge_color in colors: colors.remove(edge_color)
        else: next_node=adjacent
    if next_node!=None:
        KempeProcess(G,node,next_node,choice(list(colors)),conflict_dictionary)

def preColoring(G,D): #Pre-coloring with a greedy algorithm
    for e in G.edges(): G[e[0]][e[1]]=G[e[1]][e[0]]=None
    for e in G.edges():
        available_colors=set(range(D))
        available_colors-=set(G[e[0]].values())
        available_colors-=set(G[e[1]].values())
        if available_colors==set(): G[e[0]][e[1]]=G[e[1]][e[0]]=randint(0,D-1)
        else: G[e[0]][e[1]]=G[e[1]][e[0]]=choice(list(available_colors))

```

```

#def preColoring(G,D): #Random pre-coloring
#   for e in G.edges(): G[e[0]][e[1]]=G[e[1]][e[0]]=randint(0,D-1)

def heuristic(G,D,repetition_limit):
    repetitionCounter=0
    conflict_dictionary=createConflictDictionary(G,D)
    previous=current=totalNumberOfConflicts(conflict_dictionary)
    while previous>0:
        highest_conflict_level=maxConflictLevel(conflict_dictionary)
        node=choice(list(conflict_dictionary[highest_conflict_level]))
        KempeStart(G,D,node,conflict_dictionary)
        current=totalNumberOfConflicts(conflict_dictionary)
        if current==0: return True
        if current>=previous:
            repetitionCounter+=1
            if repetitionCounter>repetition_limit: return False
        else: repetitionCounter=0
        previous=min(previous,current)
    return True

def applyHeuristic(G,D,repetition_limit,iteration_limit):
    preColoring(G,D)
    number_of_iterations=1
    while not heuristic(G,D,repetition_limit):
        if number_of_iterations>iteration_limit: break
        preColoring(G,D)
        number_of_iterations+=1
    print "Number of iterations:",number_of_iterations
    print "Edge-coloring successful:",checkEdgeColoring(G,D)

### Example ###
repetition_limit=iteration_limit=50
NUMBER_OF_VERTICES=50000
DEGREE=3
G=nx.random_regular_graph(DEGREE,NUMBER_OF_VERTICES)
applyHeuristic(G,DEGREE,repetition_limit,iteration_limit)

```

9. Testing a conjecture on odd graphs

Odd graphs, O_k with $k \geq 2$ an integer, are defined in the following way: The vertices correspond to the $(k-1)$ -subsets of a $(2k-1)$ -set, and two vertices are adjacent if their corresponding subsets are disjoint. It is conjectured (Biggs [1], Fiorini and Wilson [3])

that odd graphs are of class one except for $k = 3$ (the Petersen graph) and k a power of two (when the graph has an odd number of vertices, implying that it is trivially of class two). In their book, Fiorini and Wilson comment results for $k \leq 8$. We used the *CVD* algorithm to test that the conjecture is true for $k \leq 11$.

10. Conclusions and future directions

We have seen that random cubic and Δ -regular graphs can be edge-colored with empirical efficiency by a relatively simple heuristic algorithm (with simple data structures).

Many combinatorial problems, for instance logical circuit problems (Fiol [2]), can be reduced to 3-edge-coloring of cubic graphs or edge-coloring of regular graphs. Therefore, any efficient edge-coloring heuristic might be useful to solve them. Moreover, efficient edge-coloring is of practical interest in many applications.

We are currently working in the generalization of the heuristic algorithm to hypergraphs and other combinatorial structures, and testing its performance in combinatorial problems.

References

- [1] N. Biggs, Some odd graph theory, Second International Conference on Combinatorial Mathematics (New York, 1978), *Ann. New York Acad. Sci.*, New York, **319** (1979), 71–81.
- [2] M.A. Fiol, A Boolean algebra approach to the construction of snarks, in Graph Theory, Combinatorics and Applications, (eds. Y. Alavi, G. Chartrand, O.R. Oellermann, and A.J. Schwenk) John Wiley & Sons **1** (1991), 493–524.
- [3] S. Fiorini and R.J. Wilson, *Edge-Colouring of Graphs*, Pitman, 1977.
- [4] M. Gardner, Mathematical Games: Snarks, Boojums and other conjectures related to the four-color-map theorem, *Sci. Amer.*, **234** (1976), 126–130.
- [5] I. Holyer, The *NP*-completeness of edge-colouring, *SIAM J. Comput.*, **10** (1981), 718–720.
- [6] R. Isaacs, Infinite families of nontrivial trivalent graphs which are not Tait colorable, *Amer. Math. Monthly*, **82**(3) (1975), 221–239.
- [7] T.T. Lee, Y. Wan, and H. Guan, Randomized Δ -edge-coloring via quaternion of complex colors (Extended Abstract), [arXiv:1104.1852v1](https://arxiv.org/abs/1104.1852v1) [cs.DS] (2011).
- [8] R.W. Robinson and N.C. Wormald, Almost all cubic graphs are hamiltonian, *Random Structures Algorithms*, **5**(2) (1994), 363–374.