Exact Graph Coloring Algorithms of Getting Partial and All Best Solutions

Jianding Guo, Laurent Moalic, Jean-Noel Martin, Alexandre Caminada
{*jianding.guo, laurent.moalic, jean-noel.martin, alexandre.caminada*}*@utbm.fr*
Univ. Bourgogne Franche-Comté, UTBM, OPERA, F-90100 Belfort, France

Abstract

Designing effective exact algorithms for graph coloring problem is still an interesting topic. Instead of getting only one best solution, two exact graph coloring algorithms, PexaCol (Partial best solutions Exact graph Coloring algorithm) and AexaCol (All best solutions Exact graph Coloring algorithm), have been proposed, which are able to obtain a best solution subset and all best solutions respectively. Based on TexaCol (Total solutions Exact graph Coloring algorithm) which is capable of getting all coloring solution subsets for each subgraph, these two algorithms utilize the backtracking method, in which they only choose the best solution subset each step to continue the coloring until partial or all best solutions are obtained. The result analysis shows that PexaCol and AexaCol can deal with larger graphs than TexaCol and especially, AexaCol runs much faster than TexaCol and the solver Gurobi to get all best solutions.

Graph coloring problem is a well-known traditional NP-complete problem, which has been extensively researched and has great significance to improve the efficiency for some industrial applications. Although there have been a lot of works on it up to now, no methods can make sure of obtaining the best solution exactly for large graphs (e.g., graphs with several thousand nodes) in an acceptable time. On the other hand, instead of getting only one best solution, very few works engage in getting multiple or all best coloring solutions which are obviously much difficult. Therefore, it is still necessary to explore new methods of graph coloring from different perspectives.

Generally, there are two kinds of methods for coloring a graph: the exact methods and the heuristic methods. The exact methods are capable of attaining the best solution for a given graph while the graph's size is small (Malaguti, Monaci, and Toth 2011; Mehrotra and Trick 1996), however, it is really limited to solve practical problems which are often modeled as larger graphs. The heuristic methods can deal with much larger graphs, nevertheless, they often cannot get the best solution. So far, in order to solve graph coloring problem, a lot of heuristic algorithms have been proposed while there are very few exact algorithms.

There have already been some studies on graph coloring by analyzing the graph structure or by decomposing the graphs. In (Rao 2004), the graph coloring is conducted using the split decomposition tree, in which a graph is recursively partitioned into smaller graphs until they cannot be split anymore. Then, after coloring the prime graphs which cannot be split, the solutions are combined gradually to get the solutions for all the graph. In (Bhasker and Samad 1991), the authors research the clique-partitioning for a graph based on the principle that the graph coloring and the clique-partitioning are equivalent to some extent. Two methods are presented to partition cliques, which perform better in runtime than some efficient graph coloring algorithms. In (Lucet, Mendes, and Moukrim 2006), an exact graph coloring algorithm is proposed by linearly decomposing a graph, which can run faster than other exact algorithms when the linearwidth is small. The graph is dynamically decomposed into subgraphs and the corresponding boundaries between these subgraphs. Then the coloring results can be obtained by analyzing different coloring cases from these boundaries.

Moreover, there are plenty of papers that work on the chromatic polynomial and the number of best solutions. In (Read 1968), the traditional method called the deletion-contraction is presented to get the chromatic polynomial which utilizes the characteristic of chromatic polynomial to do the operations to the graph. In (Lin 1993), an approximation algorithm is proposed to calculate the chromatic polynomial after obtaining its upper bound and lower bound, which has good performance in time complexity.

Our work is based on TexaCol in our previous work (Martin 2010), which can get all graph coloring solutions as well as the chromatic polynomial. TexaCol is a graph-structure-based algorithm, in which the graph is decomposed into maximal cliques, and the relationship between these maximal cliques is analyzed to get all coloring solutions. We improve this work in this paper to propose two exact graph coloring algorithms, which can get multiple best solutions or all best solutions. Our contributions are as follows:

- Based on TexaCol, an algorithm called PexaCol has been designed to get partial best solutions (see section PexaCol). Each step, instead of dealing with all solution subsets, PexaCol utilizes backtracking method to choose only the best solution subset for subgraphs to calculate until a best solution subset for all the graph is obtained.
- Based on PexaCol, we proposed another algorithm called AexaCol which is capable of getting all best solutions for

a graph (see section AexaCol). Different from PexaCol, this algorithm will not stop the calculation until all best solutions are acquired.

- By calculating a lot of graph instances, the performance of these algorithms has been evaluated (see section Result analysis). Result analysis shows that PexaCol and AexaCol are able to deal with larger graphs and can run faster than TexaCol. In addition, AexaCol can run much faster than the famous solver Gurobi in getting all best solutions.

Preliminaries

The vertex coloring of a graph G is a proper coloring such that all nodes are colored and all neighboring nodes are colored differently. In this paper, each proper coloring solution is called one solution and all proper coloring solutions are called all solutions. Partial solutions is defined as a proper coloring solution subset and all of these subsets compose all solutions. If at most k colors are used for the proper coloring, it is called proper k-coloring. The chromatic number is k if at least k colors are used for the proper coloring. If the chromatic number is k, each proper k-coloring solution is called one best solution. All best solutions is the set of all proper k-coloring solutions when k is the chromatic number. If not all but a part of the best solutions are found, we call that partial best solutions, i.e., a best solution subset.

We use a specific data structure called column to deal with the coloring solutions. Given the coloring sequence, a column denotes a proper coloring solution subset and they are equivalent in the aftermentioned part. All this kind of columns compose all graph coloring solutions. In each column, each node's coloring constraint is indicated, containing either the set of nodes which should have different colors or the same color with this node. The minus sign means two nodes have the same color. Specially, if a new node can have the same color with some colored nodes, we find the first colored node among them and add a minus sign before it as the coloring constraint of this new node. Given the number of colors k, a factor representing the number of optional colors for each node, can be obtained from each node's coloring constraint. The chromatic polynomial for a column, indicating the number of solutions included in this column, is the product of all nodes' factors. Then the chromatic polynomial for the graph is the sum of all column's chromatic polynomials. The number of colors for a column equals the maximal number of nodes in all nodes' coloring constraint plus 1. While coloring, we gradually add an uncolored node's coloring constraint to the column until all nodes are colored.

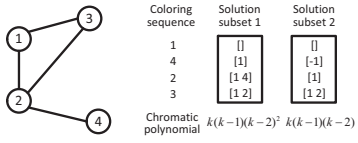

Figure 1: Example of the coloring data structure

In Figure 1, the coloring result for a graph with 4 nodes is shown, which contains two columns. All coloring solutions are included in these two columns and each column is a coloring solution subset. The coloring constraint for each node is written in the corresponding bracket. For instance, in the first column, the constraint of the node 2 is [1 4], that means the node 2 is colored differently with the node 1 and the node 4. In the second column, the constraint of the node 4 is [-1], that means the node 4 is colored the same as the node 1. The number of colors for both columns equals 3, i.e., the chromatic number. The chromatic polynomial of both columns is indicated in the figure and each column contains 6 best solutions.

TexaCol

TexaCol is an exact algorithm for getting all proper coloring solutions, which has been proposed in [9]. Based on the fact that the graph structure has significant influence on graph coloring, this algorithm decomposes a graph G into maximal cliques and colors the graph clique by clique considering the connection between these maximal cliques. In the algorithm, all proper coloring solutions of G are partitioned into several columns according to different coloring cases. For instance, if two nodes can have the same color, this gives rise to one column, representing the case that they are colored the same; meanwhile, another column can be obtained, which means the case that they are colored differently. In this way, this algorithm can obtain all proper coloring solutions for a given graph G by listing all columns without repetition. Furthermore, given the number of colors k, the chromatic polynomial for G can be gained according to these columns.

Generally, TexaCol is composed of three steps: maximal clique decomposition, suite construction and node coloring. The first two steps, as the prerequisite for node coloring, are engaged in decomposing the input graph into maximal cliques and getting a suitable sequence of them to conduct the coloring. The third step attains all the proper coloring solutions by means of analyzing the relationship between these maximal cliques while coloring them one by one.

Although TexaCol is capable of obtaining all proper coloring solutions for a graph, its performance is extremely limited. The results show that it can only deal with the coloring for very small graphs (e.g., graphs with about 15 nodes). As the number of nodes increases, getting all coloring solutions becomes really intractable, for the traverse of all coloring cases can hardly be finished in a polynomial time. In addition, using graph coloring to solve the practical problem usually requires only the best coloring solution rather than all coloring solutions. So, instead of getting all solutions, we try to get the best solution in this paper based on TexaCol.

PexaCol

General idea

From TexaCol, we have a simple method to get the best solution. After all columns for the graph have been obtained, we can choose the columns with the minimum number of colors as the best columns by calculating their number of colors. All best solutions for graph G are included in these best columns. Considering the coloring process of TexaCol, each node is colored according to a certain coloring sequence

and while coloring a new node, we add the skeleton, i.e., the initial coloring constraint, of this node to each column of colored nodes (the concept of skeleton can be found in the subsection Key concept). By dealing with the skeleton, some new columns are obtained containing the new coloring constraint of this new node. In this way, until all nodes are colored, we get all columns for the graph. That means, in TexaCol, to get all solutions for the graph, each step we need to deal with skeleton based on each column. However, to get only the best solution for graph G, only some columns are interesting.

As an improvement, in PexaCol, given the fixed coloring sequence, each step only the best columns for the subgraphs are chosen to continue the coloring until all nodes are colored. The best column here should satisfy two conditions: it requires the minimal number of colors and it has the maximal number of nodes if the first condition has been satisfied. So, in order to choose the best column each step, two values are calculated for each column: $numColor$ and $numNode$, which indicate the number of colors and the number of nodes respectively. The process of choosing the best column is illustrated in Figure 2. Let m be the total number of columns for subgraphs with different number of nodes and N be the total number of nodes. S_i represents the column i, $i = 1, 2, ..., m$. V_j is the node's index, $j = 1, 2, ..., N$. For each column i, the number of colors is c_i, and the number of nodes is n_i, $i = 1, 2, ..., m$. The minimum number of colors required for these columns is $minColor = \{c_i | c_i \leq c_j, i \neq j\}$, $i = 1, 2, ..., m$, $j = 1, 2, ..., m$, and the columns with the minimum number of colors are included in the candidate set $\{s_i | c_i = minColor\}$, $i = 1, 2, ..., m$. Then, the best column is the column in the candidate set who has the maximal number of nodes. If more than one columns satisfy the conditions of best column, we can choose one of them randomly.

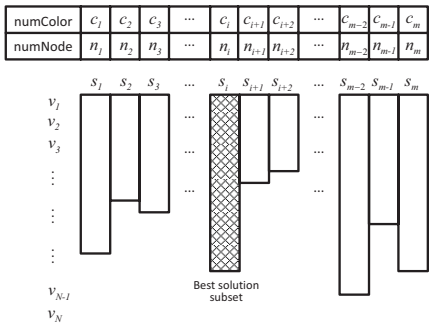

Figure 2: Choose the best column at each step

Each step we choose the best column and add a new node's coloring constraint to it according to the coloring sequence. Then new columns including this new node corresponding to different coloring cases are added to the old columns. At this time, we choose again the best column among these columns, implying that if new columns are not good, we go back and choose one among the old columns. So we call it the backtracking method. The reason why this method helps to get a best column for all the graph is explained as follows. Firstly, among columns of the subgraphs, the one requires the minimum number of colors are more likely to cause a best column for all the graph. Then, if two columns have the same number of colors, the column with the maximal number of nodes can accelerate the coloring of all the graph. If we choose the column with less nodes, the number of colors will be more than or equal to that with more nodes, as shown in the example in Figure 3. Note that after coloring a new node, the number of colors will stay the same or increase by 1 at most, for the worst case is to give the new node a new color. In Figure 3, there are two columns to choose: the column 1 with 3 colors and 4 nodes, and the column 2 with the same number of colors and 5 nodes. If we choose the column 1, there will be two possibilities, all of which cannot be better than column 2 itself which need only 3 colors for 5 nodes. For the case of choosing column 2, it will also be two possibilities: one with 3 colors and 6 nodes, which is much better than the old column 1 and column 2; the other one with 4 colors and 6 nodes, which is no better than that while choosing the old column 1. If we choose the column 2, it is really possible that we can get a much better new column; if the better column has not been obtained, we can also do the backtracking and choose the old column 1 to continue.

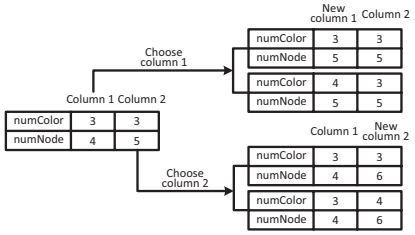

Figure 3: The possibility of choosing different columns when the number of colors is equal

Key concept

From the graph structure point of view, two essential concepts are explained below. Some relevant concepts can also be found in our previous works (Martin 2010; Guo et al. 2017).

Maximal clique: A maximal clique is a maximal set of nodes who connect with each other. If all maximal cliques are searched out, we say a graph is decomposed into the maximal cliques. It is possible that one node belongs to more than one maximal cliques.

Skeleton: All colored nodes which are adjacent to the node v are the skeleton of v. It is divided into two parts: the first part contains the maximal number of nodes in the skeleton which are adjacent to one another, called layer 1; the second part, called layer 2, includes all the rest of nodes in the skeleton. Each layer is written in a bracket. Skeleton is the initial coloring constraint for a node, which reflects the connection between a node to color and all nodes colored, and it also reflects the connection between maximal cliques.

Algorithm description

Based on TexaCol, the main flow graph for PexaCol is shown in Figure 4. It also contains three steps: maximal clique decomposition, suite construction and node coloring. Note that even the first two steps and the function of treating the skeleton are the same as that in TexaCol, PexaCol utilizes the backtracking method to do the node coloring, in which only the best column for subgraphs rather than each column is chosen out to continue the skeleton treating each step. The functions are explained in detail as follows.

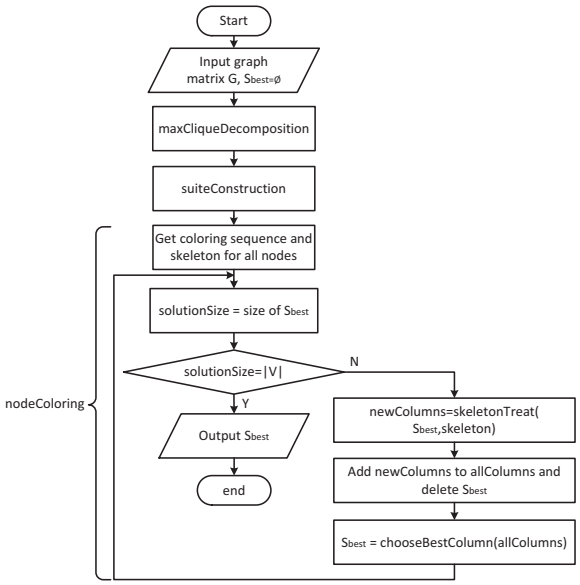

Figure 4: Flow graph of the graph coloring algorithm

maxCliquesDecomposition(): This step reflects the basic idea of coloring based on graph structure, which decomposes a graph into maximal cliques. It can be realized by searching from each node recursively until a maximal clique has been obtained. Starting with the node with the minimum index, to simplify the search process, it is confined that only the nodes whose index is larger than that of the nodes added in the clique are considered as the candidate nodes.

suiteConstruction(): Suite construction aims to find a good coloring sequence for the maximal cliques, called the suite. There are two important rules to choose the cliques: maximal constraint and maximal contact (Martin 2010). The maximal constraint equals the sum of all nodes' degree in a maximal clique subtracts the number of edges in this maximal clique. The maximal contract of a maximal clique is the number of nodes in the intersection between this maximal clique and the set of all colored nodes. The first clique in the suite is chosen having the maximal constraint and then the cliques are gradually chosen having the maximal contact. The nodes' coloring sequence is determined by the sequence of their appearance in the suite.

skeletonTreat(): It is to divide the coloring cases by analyzing each node's skeleton. If there is only one layer for one node's skeleton, the new coloring constraint of this node is the same as its skeleton and no other different possibilities of coloring. If one node's skeleton has two layers, there are different possibilities of coloring. For instance, some nodes in layer 2 may have the same color with some nodes in layer 1. So in this case, by adding this node's skeleton to a column and treating it, it will generate multiple new columns, in which the new coloring constraint of this node is attained and the relevant colored nodes' coloring constraint are changed according to different coloring cases.

chooseBestColumn(): After skeleton treating, this function chooses the best column to continue (see Algorithm 1). The input is $allColumns$, denoting the old columns for subgraphs. Two indicators are calculated for each column: $numColor$ and $numNode$. Then, a best column, which has the minimum $numColor$ and has the maximal $numNode$ among the columns with the minimum $numColor$, is chosen from all columns.

Algorithm 1: Function *chooseBestColumn()*

Input: $allColumns$
Output: best column S_{best}
Initialization: $numColor \leftarrow \emptyset, numNode \leftarrow \emptyset, minColor \leftarrow 0,$ $maxNode \leftarrow 0, S_{best} \leftarrow \emptyset$.
/* Phase 1: get $numNode$ and $numColor$ */
for *each new generated column in* $allColumns$ **do**
  get the corresponding $numNode$ and $numColor$;
/* Phase2: choose the best column */
$minColor \leftarrow numColor[1]$;
$maxNode \leftarrow numNode[1]$;
$numColumns \leftarrow sizeof(allColumns)$;
for *i from 1 to* $numColumns$ **do**
  if $numColor[i+1] < minColor$ /* If columns with less colors have been found */
  then
    $S_{best} \leftarrow allColumns[i+1]$;
    $minColor \leftarrow numColor[i+1]$;
    $maxNode \leftarrow numNode[i+1]$; /* Update $minColor$ and $maxNode$ */
  else if $numColor[i+1] = minColor$ **then**
    if $numNode[i+1] > maxNode$/* Choose the column with more nodes */
    then
      $S_{best} \leftarrow allColumns[i+1]$;
      $maxNode \leftarrow numNode[i+1]$;

Example

An example is given to illustrate how to choose the best column each step. The input graph and its adjacent upper triangular matrix are shown in Figure 5. The suite of maximal cliques and the coloring sequence are shown in Table 1. We color the vertices one by one according to the coloring sequence. The column for the first 5 nodes can simply equal their skeletons, which have only one layer.

As it is shown in Figure 6, after the vertex 4 has been colored, we get two columns (columns at left side), representing the case that the vertex 3 and the vertex 5 have the same color and different colors respectively. Then, after calculat-

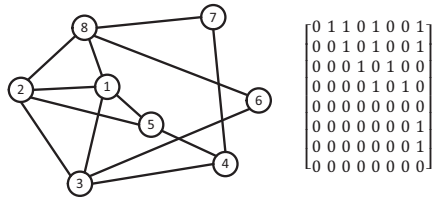

Figure 5: The input graph and its adjacent matrix

Table 1: Basic information for coloring the input graph

suite	coloring sequence	skeleton	column for the first 5 nodes
{1,2,3}	1	[ ]	[ ]
{1,2,8}	2	[1]	[1]
{1,2,5}	3	[2 1]	[2 1]
{3,4}	8	[2 1]	[2 1]
{4,5}	5	[2 1]	[2 1]
{3,6}	4	[3] [5]	
{6,8}	6	[3] [8]	
{7,8}	7	[8] [4]	
{4,7}			

ing each column's $numColor$ and $numNode$, the column with 3 colors is chosen, because it has the minimum number of colors. Based on this column, by treating the skeleton of the node 6 (in the dashed rectangle at the left side), two new columns are obtained (columns in the middle). Then, $numColor$ and $numNode$ for these new columns are calculated. Because the column with 3 colors and 7 nodes is obviously better, it is chosen to continue. At the end, the best column for all the graph is obtained, which only requires 3 colors, i.e., the chromatic number (columns at right side).

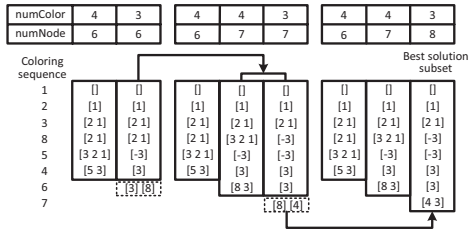

Figure 6: Example of choosing the best column

AexaCol

Instead of getting only a part of best solutions, AexaCol is able to get all best solutions for a given graph. Here the number of all best solutions equals the value of the chromatic polynomial when k is the chromatic number. For PexaCol, it will stop while getting a best column in which all nodes are colored, so only partial best solutions are obtained. However, to get all best solutions, we need to obtain all columns whose $minColor$ equals the chromatic number.

It can be realized in this way. After obtaining the first best column for the graph, we know the chromatic number. Then, all columns whose $numColor$ is larger than the chromatic number will be deleted, because it is sure that they cannot be the best columns. Each step before choosing a best column to continue, all columns whose $numColor$ is larger than the chromatic number are also deleted. Until no columns left in $allColumns$, all best solutions are obtained. In the example in section PexaCol, only one column has the minimum number of colors, so this column also contains all best solutions.

Result analysis

The result analysis is conducted in this section. To evaluate the performance of our algorithms, we mainly focus on the number of best solutions achieved, the number of columns and the runtime. The experiment has been implemented under the system Ubuntu 14.4 on a computer with CPU Intel Core(TM) i7-4790 (3.60 GHZ, 3.60 GHZ) and RAM 8 Go.

Firstly, as there are really few algorithms who are engaged in getting all best solutions, our algorithm AexaCol is compared with the famous solver Gurobi in getting all best coloring solutions (Gurobi Optimization 2017). Both Gurobi and our algorithm are implemented in C++. We let Gurobi to get the chromatic number first with the objective of using the minimal number of colors and the constraints that each node has one color and all adjacent nodes are colored differently. Then we use it to get all best solutions based on the same objective and constraints by setting the model parameter "PoolSearchMode". As Gurobi cannot finish most of the small DIMACS graphs, in order to draw the comparison, a lot of random graphs are used. All graphs whose name begins with "gr_" are created randomly by ourselves, while all other graphs are DIMACS benchmark graphs. The result is shown in Table 2. V denotes the number of nodes and E denotes the number of edges. χ is the chromatic number. N_a is the number of all best coloring solutions, and t is the runtime, whose unit is second. While Gurobi and AexaCol can both get all best solutions for the instances, the comparison is drawn on runtime. As shown in the table, AexaCol can run much faster than Gurobi. For all these graphs, AexaCol can get all best coloring solutions very quickly. However, Gurobi costs a lot of time. Taking gr_n23_e80 for example, Gurobi's runtime is 193190 times of that with AexaCol.

Table 2: Comparison between AexaCol and Gurobi

graph name	V	E	χ	N_a	$t(s)$	
					Gurobi	AexaCol
queen5_5.col	25	160	5	240	0.303	0.011
gr_n15_e25	15	25	4	147456	102.335	0.003
gr_n15_e30	15	30	4	36864	7.111	0.002
gr_n20_e87	20	87	5	2880	0.362	0.006
gr_n17_e20	17	20	3	52488	15.442	0.002
gr_n18_e35	18	35	4	24576	4.428	0.002
gr_n18_e64	18	64	5	61919	24.417	0.026
gr_n17_e65	17	65	5	127200	101.460	0.017
gr_n15_e28	15	28	4	150523	123.103	0.002
gr_n19_e47	19	47	4	24576	4.449	0.002
gr_n19_e49	19	49	4	18432	2.985	0.003
gr_n22_e80	22	80	5	150000	124.792	0.004
gr_n23_e80	23	80	5	300000	579.571	0.003

Table 3: Comparison between TexaCol, AexaCol and PexaCol

graph name	V	E	χ	TexaCol				AexaCol				PexaCol			
				N_c	N_a	N_s	$t(s)$	N_c	N_a	N_s	$t(s)$	N_c	N_a	N_s	$t(s)$
myciel3.col	11	20	4	4	12480	967	0.033	4	12480	431	0.01	4	384	70	0.004
myciel4.col	23	71	5	\	\	\	\	5	2.85e+09	1023580	29.012	5	7.16e+06	21309	0.413
queen5_5.col	25	160	5	\	\	\	\	5	240	57	0.011	5	240	57	0.01
queen6_6.col	36	290	7	\	\	\	\	\	\	\	\	7	20160	18152	0.547
queen7_7.col	49	476	7	\	\	\	\	7	20160	174055	4.266	7	20160	174055	4.254
gr_n15_e51	15	51	5	5	42600	19697	0.276	5	42600	951	0.017	5	1800	51	0.002
gr_n16_e56	16	56	5	5	15840	708	0.013	5	15840	27	0.002	5	15840	27	0.002
gr_n17_e62	17	62	5	5	9000	9072	0.206	5	9000	77	0.003	5	9000	77	0.002
gr_n18_e91	18	91	7	7	5.38e+06	26220	0.761	7	5.38e+06	718	0.02	7	5.38e+06	718	0.018
gr_n19_e86	19	86	5	\	\	\	\	5	360	100	0.005	5	360	100	0.003
gr_n20_e95	20	95	6	\	\	\	\	6	1.73e+06	4918	0.089	6	47520	124	0.006
gr_n21_e105	21	105	6	\	\	\	\	6	634320	5963	0.11	6	23760	183	0.007
gr_n22_e116	22	116	6	\	\	\	\	6	166320	4796	0.086	6	720	41	0.006
gr_n23_e125	23	125	6	\	\	\	\	6	89280	3099	0.057	6	4320	149	0.007
gr_n24_e133	24	133	6	\	\	\	\	6	103680	3089	0.058	6	2880	143	0.008
gr_n25_e149	25	149	6	\	\	\	\	6	2880	726	0.02	6	1440	212	0.013
gr_n26_e166	26	166	6	\	\	\	\	6	720	346	0.019	6	720	111	0.013
gr_n27_e180	27	180	7	\	\	\	\	7	6.43e+07	51309	1.212	7	221760	327	0.02
gr_n28_e193	28	193	7	\	\	\	\	7	1.14e+08	85030	1.974	7	292320	394	0.024
gr_n29_e212	29	212	7	\	\	\	\	7	6.86e+06	10676	0.259	7	20160	910	0.044
gr_n30_e229	30	229	7	\	\	\	\	7	171360	2435	0.076	7	30240	866	0.047
gr_n31_e244	31	244	7	\	\	\	\	7	20160	1075	0.057	7	20160	1075	0.056
gr_n32_e259	32	259	7	\	\	\	\	7	40320	1127	0.063	7	40320	1127	0.062
gr_n33_e277	33	277	8	\	\	\	\	8	3.93e+09	449094	11.873	8	161280	1031	0.067
gr_n34_e294	34	294	8	\	\	\	\	8	9.86e+08	166416	4.289	8	403200	958	0.085
gr_n35_e291	35	291	8	\	\	\	\	\	\	\	\	8	1.57e+07	81431	1.734
gr_n36_e308	36	308	8	\	\	\	\	\	\	\	\	8	806400	53079	1.183
gr_n50_e206	50	206	8	\	\	\	\	8	1.56e+29	700830	32.232	8	2.60e+25	350	0.036
gr_n58_e518	58	518	9	\	\	\	\	\	\	\	\	9	1.14e+17	43448	1.862
gr_n70_e194	70	194	7	\	\	\	\	7	1.775e+42	8828	0.277	7	3.84e+41	1971	0.195

Furthermore, the comparison has been conducted between TexaCol, AexaCol and PexaCol, as shown in Table 3. N_c is the minimum number of colors required for the best columns in each algorithm. Because the number of all best solutions N_a can be really large, its decimal is rounded off. For example, for the DIMACS graph myciel4.col, N_a is 2.84566e+09. After being rounded off, it is written as 2.85e+09. N_s represents the number of columns in $allColumns$ when the algorithm is finished. In our result, it has counted the best columns for the subgraph each step. That means, we do not delete them in the data structure after having chosen and treated them. Generally, it will consume more computer's memory with larger N_s. The backslash means the result cannot be obtained.

From the table, it can be seen that TexaCol is unable to get the solutions for a lot of graphs, for it calculates all columns of each subgraph rather than the best column, which has very high computational complexity. Here the space complexity is the key problem. For TexaCol, it often crashes due to the memory shortage because it has extremely large N_s. As shown in the table, even for very small graphs, TexaCol has larger N_s than two other algorithms and PexaCol always has the least N_s. So TexaCol fails to get the result for a lot of graphs and AexaCol fails for few graphs while PexaCol can get the result. On the other hand, N_s reflects the range of choosing the best column, signifying that the less N_s, the less runtime. In general, PexaCol runs much faster than AexaCol and AexaCol runs much faster than TexaCol. For instance, for the graph gr_n15_e51, PexaCol's runtime is about 12% of AexaCol's and 0.7% of TexaCol's. Moreover, for TexaCol and AexaCol, they are always able to get all best solutions while PexaCol can only get a part of best solutions, which shows that finding more best solutions costs more time. Taking myciel3.col for example, PexaCol runs 25 times faster than AexaCol, but it can only get 384 best solutions, while the number of all best solutions is 12480.

Conclusion

Based on TexaCol, two algorithms, PexaCol and AexaCol, have been proposed in this paper, which are able to get partial and all best coloring solutions respectively. Instead of calculating all columns for each subgraph, these two algorithms only choose the best column to continue the calculation each step. The result analysis shows that these two algorithms run faster than TexaCol and can deal with larger graphs. Furthermore, the proposed algorithm AexaCol can run much faster than the famous solver Gurobi to get all best solutions. As future work, it is still interesting to find good mechanisms to get all best coloring solutions on the basis of these graph-structure-based methods.

References

Bhasker, J., and Samad, T. 1991. The clique-partitioning problem. *Computers & Mathematics with Applications* 22(6):1–11.

Guo, J.; Moalic, L.; Martin, J.-N.; and Caminada, A. 2017. Cluster resource assignment algorithm for device-to-device networks based on graph coloring. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2017 13th International*, 1700–1705. IEEE.

Gurobi Optimization, I. 2017. Gurobi optimizer reference manual.

Lin, N.-W. 1993. Approximating the chromatic polynomial of a graph. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, 200–210. Springer.

Lucet, C.; Mendes, F.; and Moukrim, A. 2006. An exact method for graph coloring. *Computers & operations research* 33(8):2189–2207.

Malaguti, E.; Monaci, M.; and Toth, P. 2011. An exact approach for the vertex coloring problem. *Discrete Optimization* 8(2):174–190.

Martin, J.-N. 2010. *No Free Lunch et recherche de solutions structurantes en coloration.(No Free Lunch and research of structuring solutions in graph coloring).* Ph.D. Dissertation, Université de technologie de Belfort-Montbéliard, France.

Mehrotra, A., and Trick, M. A. 1996. A column generation approach for graph coloring. *informs Journal on Computing* 8(4):344–354.

Rao, M. 2004. Coloring a graph using split decomposition. In *WG*, 129–141. Springer.

Read, R. C. 1968. An introduction to chromatic polynomials. *Journal of Combinatorial Theory* 4(1):52–71.