Minimal partition in planar subgraphs

Hautefeuille and Hebras

February 2022

Here is our project. We really enjoyed working on it We preferred coding the main class ourselves entirely. We participated in the tournament and finished 13^{th} in the junior competition. We had ideas to intent going further that will be mentioned but we did not have time to do so.

1 Method

From the geometrical undirected graph $G = (v, E)$, we build an undirected graph $G' = (V', E')$ with $V' = E$ and for all $(u, v) \in V'$, u and v are linked as vertices of G' if and only if u and v as edges of G cross each other. Thus we shift to a representation that is no longer spatial.

Let us notice that a minimal partition in planar sub-graphs of G induces a coloring of G', and conversely : two edges that cross in G are vertices in G' that are linked and cannot have the same color.

We sought a counter-example to show that the function that transforms G in G' does not reach the set of all graphs, bearing in mind that we could make the most of a special characteristic, but we did not find any. This would be the way to go further.

Then it is sufficient to color G'. The number of color of a coloring is the number of classes of the associated partition.

2 Coloring algorithm, complexity and data structure

Is is known that coloring a graph is an NP-complete problem, and consequently we were looking for an approximation algorithm.

We have experienced varied coloring algorithms. We have started by exploring

the NetworkX Python library and its diversity, to empirically research which algorithm is the most efficient and if it varies for specific graphs.

It appeared that in most of the case the DSATUR algorithm was the most efficient. It is also known that the main research concerning coloring algorithms consists in making the most of the special characteristics of a given subset of the set of all graphs to design an efficient algorithm. However we have not managed to find the right way into this.

Here is how this greedy algorithm proceeds.The greedy decisions are made following the following indicator : let $DSAT(v) = \textit{number of different colors adjacent to } v$ be the degree saturation of any vertex v.

1. Select v be the vertex in G with the highest DSAT among the uncolored ones. In cases of ties, choose the vertex among these with the largest degree (in the subgraph induced by the uncolored vertices).
2. Assign v to the lowest color label not being used by any of its neighbors.
3. If all vertices have been colored, the algorithm is over; otherwise return to Step 1.

For the sake of quickness, we chose Java to write our final code after trials with Python (too slow for the large instances). One of the challenge was the memory usage. When the initial geometrical graph G has 75.000 vertices (in the case of the biggest instance), the transformed graph G' has a maximum of $75.000 \times 7.5000 = 5.625.000$ edges, in fact it was around 1.500.000 edges. Using adjacency lists, this implies storing 3.000.000 integers (the sum of the degrees equals twice the number of edges) in memory, which is possible with computers with 16 GB of RAM, because Java uses a 4 bytes representation for its primitive integers ($4 \times 3.000.000 = 12GB \leq 16GB$).

Thus, to construct the adjacency list, we could not use an dynamic *Array* of *ArrayList* because this would have used the type *Integer* (and not *int*), which are stored on 16 bytes.

Consequently, we first computed the degree of each node in the transformed graph G' to create an array of arrays with correct lengths. At the end, the creation of the transformed graph G' runs in $O(m^2)$ where m is the number of edges in the original graph G. Indeed, for each pair vertices in G, we need to determine whether they cross each other or not.

Because of this initial quadratic complexity, we uses a basic implementation of DSatur (useless to do better with binary heaps), which is also quadratic in time in the number of nodes in G'.

To avoid computing the surrounding colors of each vertice in G' at each step of DSatur, we store them in a list of set (one set for each vertex), which is efficient for a fast insertion the new colors.

Overall, our implementation runs in $O(m^2)$ where m is the number of edges of G, the initial geometrical graph.

3 Experimental results and run-time

In the file submitted there is a file *CG:SHOP-solutions* that contains the 225 solutions we uploaded for the competition and obtained with a 16GB RAM PC. They were all validated by the verification program of the competition.

Here are the results obtained with a 8GB RAM PC on the instances given for the programming project when running the program with parameter *verbose = false*.

- file: small.json (instance 0 / 11)
  Run time graph transforming : 0.0
  Run time DSatur : 0.001
  Number of classes in the partition : 2
- file: vispecn2518.instance.json (instance 1 / 11)
  Run time graph transforming : 0.584
  Run time DSatur : 0.421
  Number of classes in the partition : 67
- file: rsqrp4641.instance.json (instance 2 / 11)
  Run time graph transforming : 1.115
  Run time DSatur : 1.86
  Number of classes in the partition : 91
- file: vispecn26025.instance.json (instance 3 / 11)
  Run time graph transforming : 24.574
  Run time DSatur : 37.765
  Number of classes in the partition : 276
- file: rvisp5013.instance.json (instance 4 / 11)
  Run time graph transforming : 0.844
  Run time DSatur : 0.417
  Number of classes in the partition : 58
- file: sqrpecn30017.instance.json (instance 5 / 11)
  Run time graph transforming : 87.579
  Run time DSatur : 305.431
  Number of classes in the partition : 434

- file: rvispecn2615.instance.json (instance 6 / 11)
  Run time graph transforming : 1.463
  Run time DSatur : 0.482
  Number of classes in the partition : 45

- file: visp26405.instance.json (instance 7 / 11)
  Run time graph transforming : 18.288
  Run time DSatur : 27.325
  Number of classes in the partition : 98

- file: sqrp7730.instance.json (instance 8 / 11)
  Run time graph transforming : 4.663
  Run time DSatur : 11.382
  Number of classes in the partition : 115

- file: sqrpecn3020.instance.json (instance 9 / 11)
  Run time graph transforming : 0.669
  Run time DSatur : 1.234
  Number of classes in the partition : 135

- file: reecn3382.instance.json (instance 10 / 11)
  Run time graph transforming : 0.862
  Run time DSatur : 0.858
  Number of classes in the partition : 94
  Over