GRAPH THEORY
[7]

Graph Coloring

Documents are here:

https://www-l2ti.univ-paris13.fr/~viennet/ens/2024-USTH-Graphs

Emmanuel Viennet
emmanuel.viennet@univ-paris13.fr

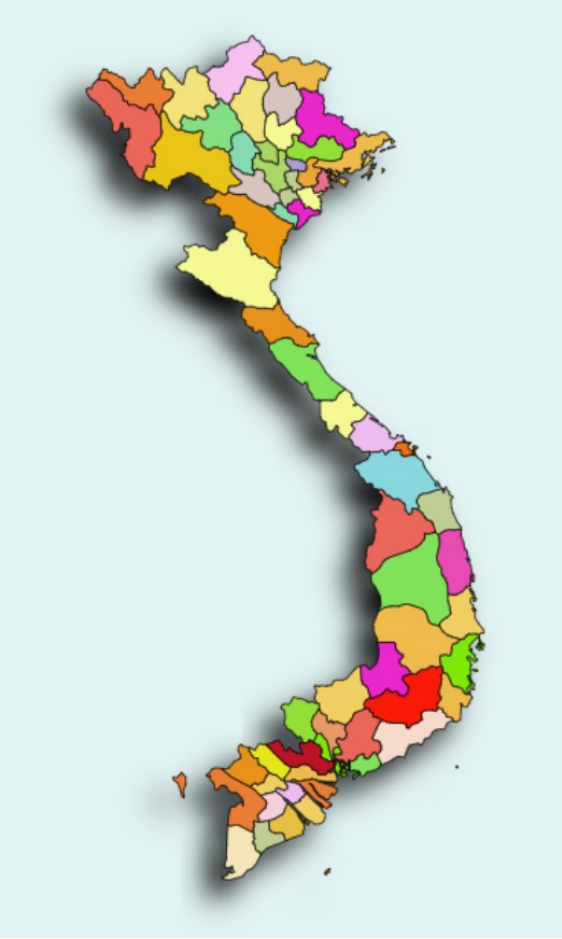
USTH

Université Sorbonne Paris Nord

Graph Coloring

- Coloring maps and graphs
- Chromatic number
- Applications of graph coloring

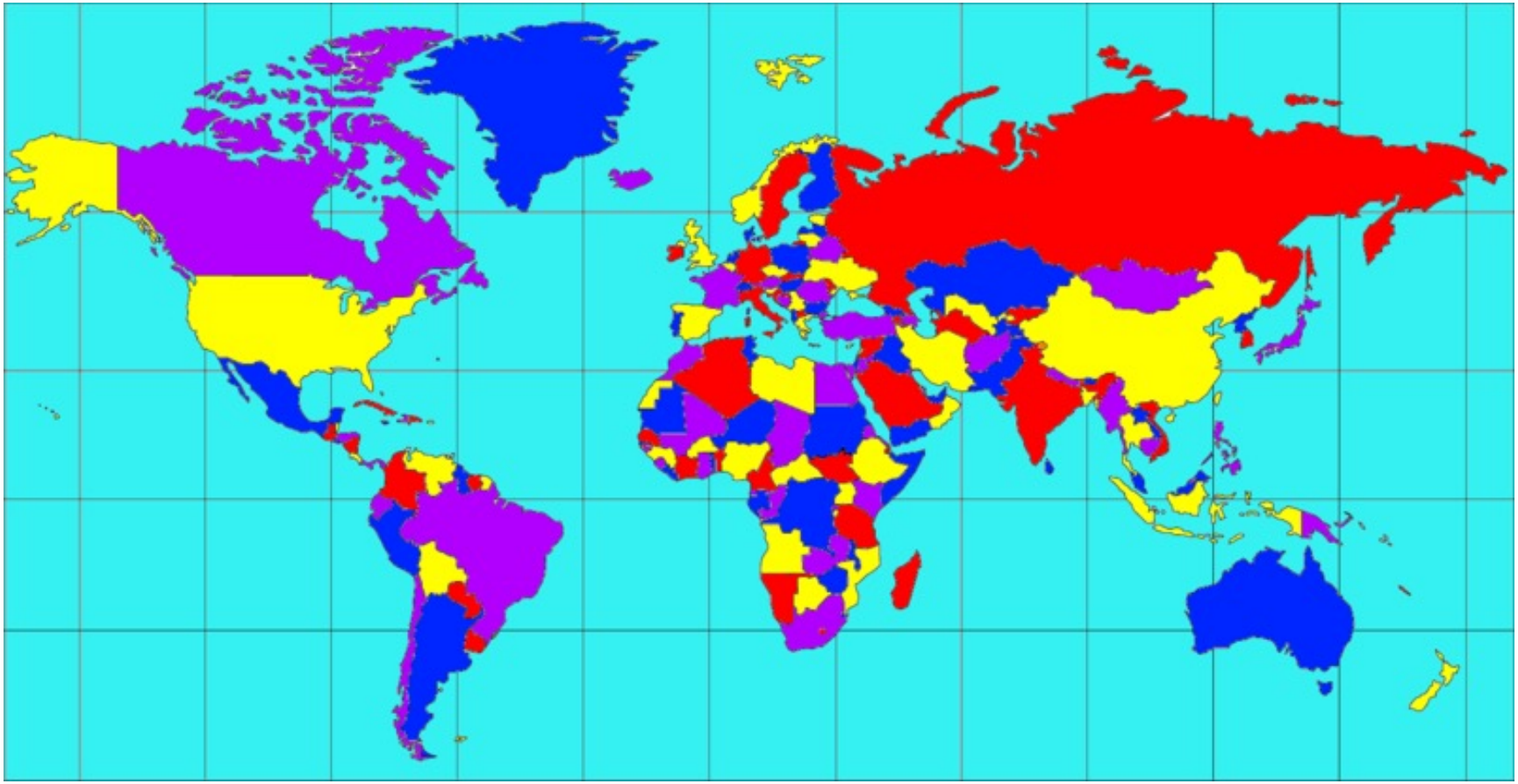
Map Coloring



Is it possible to use less colors ?

(such that no adjacent regions share the same color !)

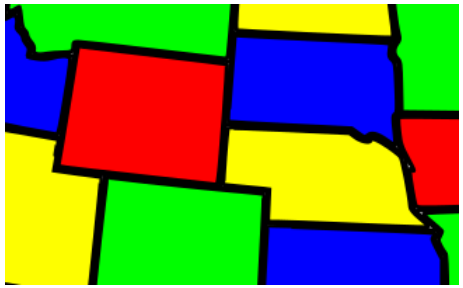
Map coloring: 4 colors are enough



wikipedia

Map coloring

Color a map such that two regions with a common border are assigned different colors.



Each map can be represented by a graph:

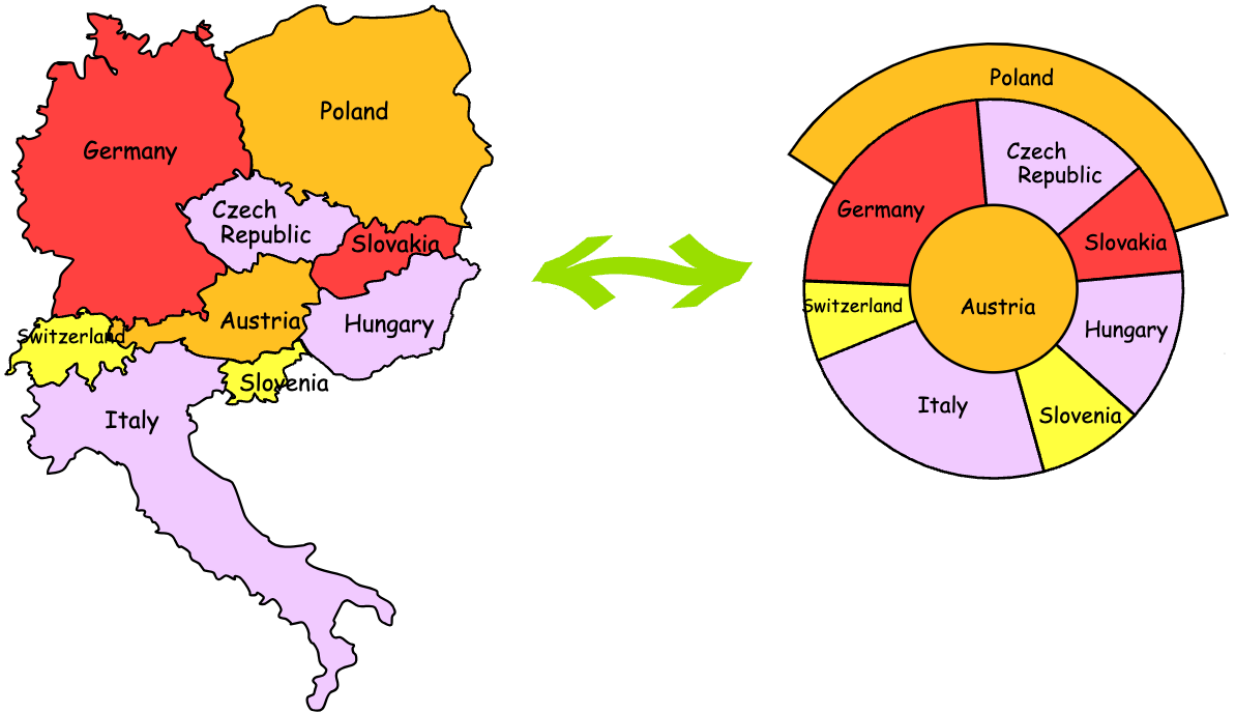
- Each region of the map is represented by a vertex;
- Edges connect two vertices if the regions represented by these vertices have a common border.

The resulting graph is called the *dual graph* of the map.

Map coloring

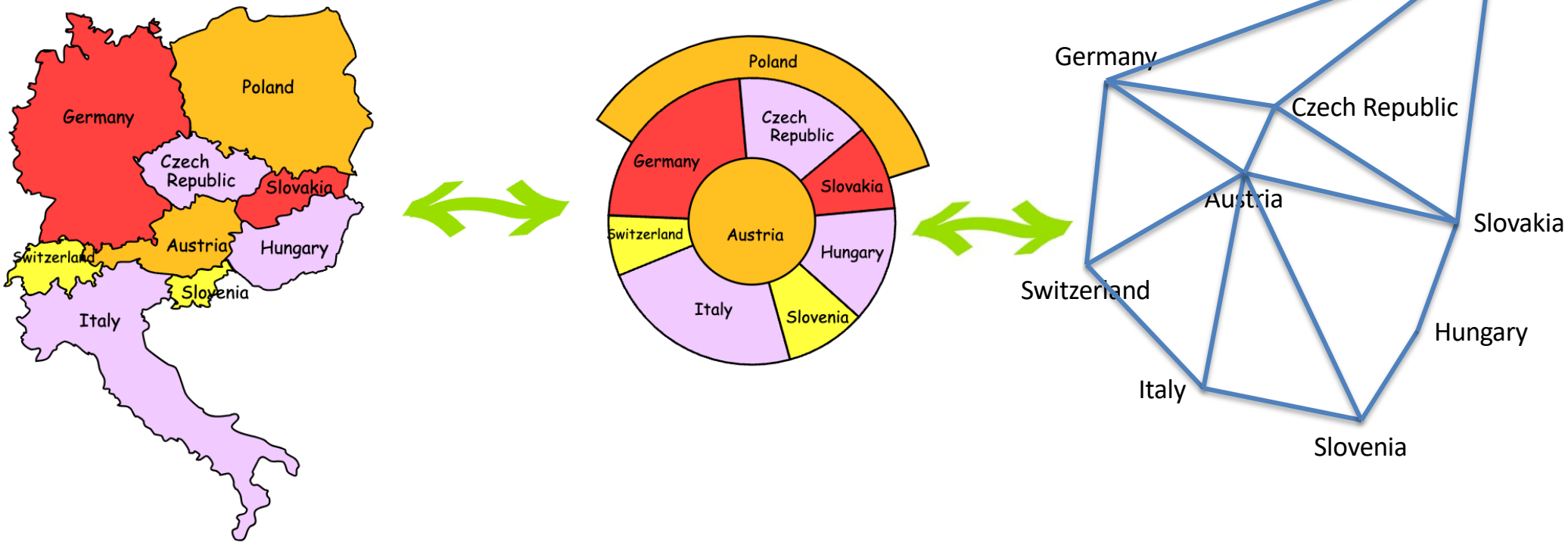
https://www.mathsisfun.com/activity/coloring.html

Map coloring



Can you see the similarity between these two diagrams?

Map coloring



C

Planar graph

Map coloring

How many colors do we need to color the regions on a map, so that adjacent regions have different colors?

Some maps require four colors. For example:

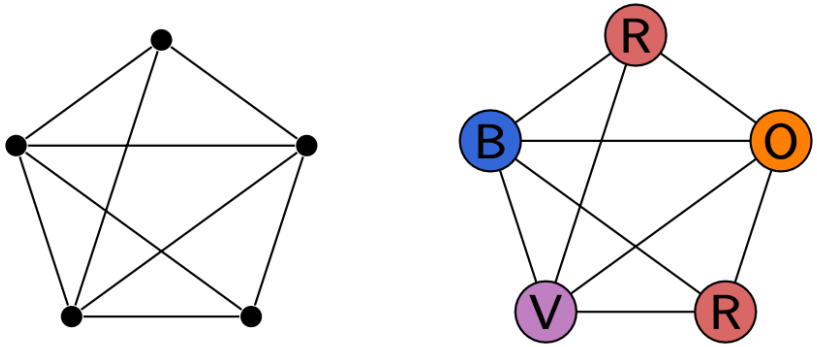
Whatever color Kentucky has, its 7 neighboring states can't be colored using only 2 more colors; they form an odd cycle.

Are four colors always enough? That's a much harder problem!

Vertex coloring

Generalizing this problem, a (vertex) coloring of G assigns every vertex of G a color; formally, it is a function $f : V(G) \to R$, where R is a set of colors.

A **proper (vertex) coloring** of G is a coloring that gives adjacent vertices different colors: $vw \in E(G) \implies f(v) \neq f(w)$.



We say that G is k-**colorable** if it has a proper coloring with k colors. The **chromatic number** $\chi(G)$ is the least k such that G is k-colorable.

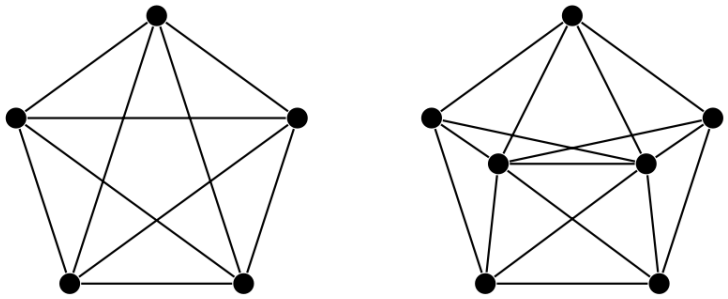
The four-color theorem

The general vertex coloring problem has many applications: register allocation in compilers, scheduling problems, etc.

Map coloring is equivalent to coloring a planar graph.

Theorem (Appel–Haken, 1976). Every planar graph is 4-colorable.

There is some intuition for this: the smallest graph that's not 4-colorable is K_5, and K_5 is not planar.



Was posed as a conjecture in the 1850s. Finally proved in 1976 with the help of computers.

The four-color theorem - *historical note*

F. Guthrie first conjectured the theorem in **1852**. In 1878, Cayley wrote the first paper on the conjecture.

Fallacious proofs were given independently by Kempe (1879) and Tait (1880). Kempe's proof was accepted for a decade until Heawood showed an error using a map with 18 faces.

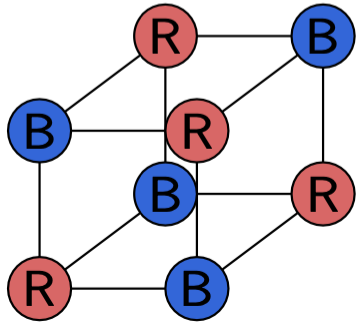
This result was finally obtained by **Appel and Haken (1977)**, who constructed a computer-assisted proof that four colors were sufficient. However, because part of the proof consisted of an exhaustive analysis of many discrete cases **by a computer**, *some mathematicians did not accept it.*

In December 2004, G. Gonthier of Microsoft Research in Cambridge, England (working with B. Werner of INRIA in France) announced that they had verified the Robertson *et al.* proof by formulating the problem in the equational logic program Coq and *confirming the validity of each of its steps* (Devlin 2005, Knight 2005).

Bipartite and 2-colorable graphs

Bipartite graphs are exactly the 2-colorable graphs. Two ways to say the same thing:

- "$V(G)$ can be partitioned into $A \cup B$ such that all edges have one endpoint in A and one endpoint in B"
- "The vertices of G can be colored red and blue such that all edges have one red endpoint and one blue endpoint."



Ckecking bipartite graphs

We can check if a graph is bipartite (2-colorable) by trying to color it:

1. Color an arbitrary vertex red (because the colors are equivalent).
2. Color all its neighbors blue (because they cannot be red).
3. Color all of the blue vertices' neighbors red, and so on.

Eventually, we will have successfully colored everything, or this strategy will tell us to color a vertex blue when it has already been colored red (or red when it has already been colored blue).

If that happens, there is no 2-coloring, and the graph has an odd cycle.

Checking if a graph is k-colorable for $k \geq 3$ is very hard! There is no efficient algorithm known.

Bounds and coloring algorithms

If it's hard to determine the chromatic number of a graph, what can we do?

We can try to prove upper and lower bounds on the chromatic number of a graph.

Today, we will focus on upper bounds on the chromatic number. The best kind of upper bound is a constructive upper bound. Constructive upper bounds will give us an algorithm to color the graph with **some** number of colors, even though it may not be the best number.

For example, for any n-vertex graph G, its chromatic number $\chi(G)$ satisfies $\chi(G) \leq n$. This corresponds to the very simple algorithm "give each vertex its own color".

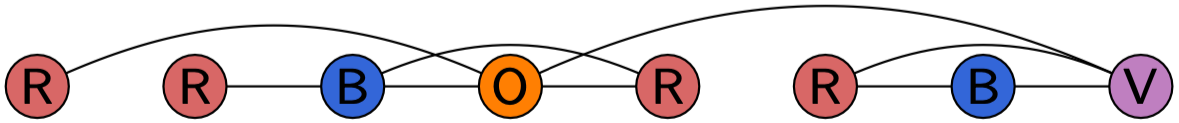
The greedy algorithm

The greedy algorithm is a slightly more intelligent coloring strategy.

Here, we assume that the vertices of a graph are ordered $v_1, v_2, \ldots, v_n$. The greedy algorithm colors v_1, then v_2, then v_3, and so on. It follows one simple rule:

- When coloring vertex v_i, use any available color[1] that was not used on any of v_i's neighbors that have been colored so far.

Here is an example. The colors we'll try are R B O V in order.



[1] For concreteness, "the first" color?

Improved bound (using greedy)

Theorem. Every graph G with maximum degree $\Delta(G)$ has chromatic number $\chi(G) \leq \Delta(G) + 1$.

Proof. This is how many colors the greedy algorithm uses, in the worst case.

If we have $\Delta(G) + 1$ colors available, then the greedy algorithm will never run out of colors. Each vertex we color has at most $\Delta(G)$ neighbors that have already been colored; even if they all get different colors, there is one color remaining. □

Improved bound (using greedy)

Theorem. Every graph G with maximum degree $\Delta(G)$ has chromatic number $\chi(G) \leq \Delta(G) + 1$.

Proof. This is how many colors the greedy algorithm uses, in the worst case.

If we have $\Delta(G) + 1$ colors available, then the greedy algorithm will never run out of colors. Each vertex we color has at most $\Delta(G)$ neighbors that have already been colored; even if they all get different colors, there is one color remaining. □

Improving greedy: DSATUR (Brélaz, 1979)

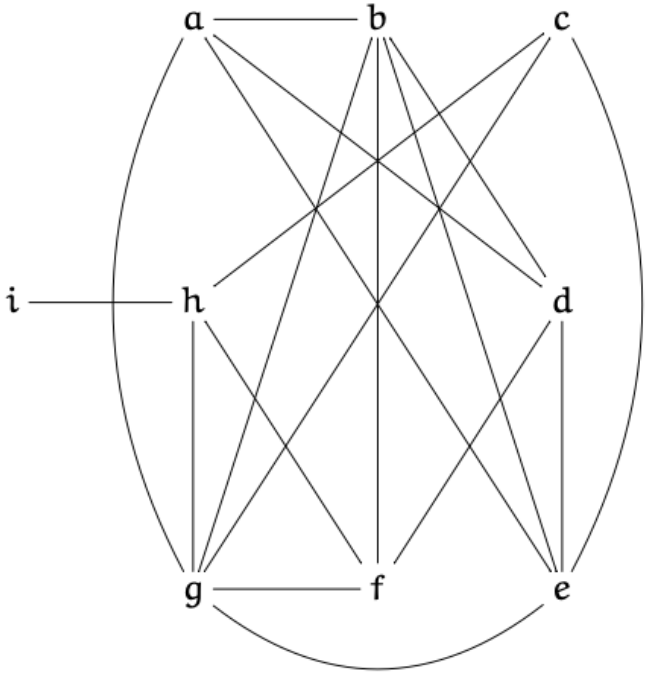
DSatur colors the vertices of a graph one after another, adding a previously unused color when needed.
Once a new vertex has been colored, the algorithm determines which of the remaining uncolored vertices has the highest number of colors in its neighborhood and colors this vertex next.
Brélaz defines this number as the **degree of saturation** of a given vertex.
The contraction of the term "degree of saturation" forms the name of the algorithm.
DSatur is a heuristic graph coloring algorithm, yet produces exact results for bipartite, cycle, and wheel graphs.

DSATUR algorithm (Brélaz)

```
while there exists uncolored nodes:
    for each uncolored node x:
        if no neighbor is colored:
            DSAT[x] = degree of x
        else:
            DSAT[x] = num. of colored neighbors

    Pick the uncolored node n with max DSAT
    (if several possibilities, choose the one with max degree)
    Color node n with the lowest possible color
```

DSATUR algorithm : example



Nodes' degrees:

$\mathbf{Som}(\mathcal{G})$	a	b	c	d	e	f	g	h	i
$d^1(x,\mathcal{G})$	4	5	3	4	5	4	6	4	1

Table with nodes sorted by growing degrees:

$\mathbf{Som}(\mathcal{G})$	g	b	e	a	d	f	h	c	i
$\mathtt{DSAT}(x)_1$	6	5	5	4	4	4	4	3	1
$\mathtt{DSAT}(x)_2$									
$\mathtt{DSAT}(x)_3$									
$\mathtt{DSAT}(x)_4$									
$\mathtt{DSAT}(x)_5$									
$\mathtt{DSAT}(x)_6$									
$\mathtt{DSAT}(x)_7$									
$\mathtt{DSAT}(x)_8$									
$\mathtt{DSAT}(x)_9$									
COULEUR									

Example from
D. Hébert, USPN

DSATUR algorithm : example

1

Max DSAT for node g, color with 1

$Som(\mathcal{G})$	g	b	e	a	d	f	h	c	i
$DSAT(x)_1$	6	5	5	4	4	4	4	3	1
$DSAT(x)_2$	■	1	1	1	4	1	1	1	1
$DSAT(x)_3$	■								
$DSAT(x)_4$	■								
$DSAT(x)_5$	■								
$DSAT(x)_6$	■								
$DSAT(x)_7$	■								
$DSAT(x)_8$	■								
$DSAT(x)_9$	■								
COULEUR	1								

2

Max DSAT for node d, color with 1

$Som(\mathcal{G})$	g	b	e	a	d	f	h	c	i
$DSAT(x)_1$	6	5	5	4	4	4	4	3	1
$DSAT(x)_2$	■	1	1	1	4	1	1	1	1
$DSAT(x)_3$	■	2	2	2	■	2	1	1	1
$DSAT(x)_4$	■				■				
$DSAT(x)_5$	■				■				
$DSAT(x)_6$	■				■				
$DSAT(x)_7$	■				■				
$DSAT(x)_8$	■				■				
$DSAT(x)_9$	■				■				
COULEUR	1				1				

DSATUR algorithm : example, cont.

3

Max DSAT+degree for node b, color with 2

Som($\mathcal{G}$)	g	b	e	a	d	f	h	c	i
DSAT(x)$_1$	6	5	5	4	4	4	4	3	1
DSAT(x)$_2$	■	1	1	1	4	1	1	1	1
DSAT(x)$_3$	■	2	2	2	■	2	1	1	1
DSAT(x)$_4$	■	■	3	3	■	3	1	1	1
DSAT(x)$_5$	■	■			■				
DSAT(x)$_6$	■	■			■				
DSAT(x)$_7$	■	■			■				
DSAT(x)$_8$	■	■			■				
DSAT(x)$_9$	■	■			■				
COULEUR	1	2			1				

4

Max DSAT for node d, color with 1

Som($\mathcal{G}$)	g	b	e	a	d	f	h	c	i
DSAT(x)$_1$	6	5	5	4	4	4	4	3	1
DSAT(x)$_2$	■	1	1	1	4	1	1	1	1
DSAT(x)$_3$	■	2	2	2	■	2	1	1	1
DSAT(x)$_4$	■				■				
DSAT(x)$_5$	■				■				
DSAT(x)$_6$	■				■				
DSAT(x)$_7$	■				■				
DSAT(x)$_8$	■				■				
DSAT(x)$_9$	■				■				
COULEUR	1				1				

DSATUR algorithm : example, cont.

Viens ensuite le sommet e qui est adjacent à g et à b ; il faut donc le colorier avec la couleur 3

Som($\mathcal{G}$)	g	b	e	a	d	f	h	c	i
DSAT(x)$_1$	6	5	5	4	4	4	4	3	1
DSAT(x)$_2$	■	1	1	1	4	1	1	1	1
DSAT(x)$_3$	■	2	2	2	■	2	1	1	1
DSAT(x)$_4$	■	■	3	3	■	3	1	1	1
DSAT(x)$_5$	■	■	■	4	■	3	1	2	1
DSAT(x)$_6$	■	■	■		■				
DSAT(x)$_7$	■	■	■		■				
DSAT(x)$_8$	■	■	■		■				
DSAT(x)$_9$	■	■	■		■				
COULEUR	1	2	3		1				

Le sommet h est adjacent à g mais pas à b ; on le colorie avec la couleur 2.

Som($\mathcal{G}$)	g	b	e	a	d	f	h	c	i
DSAT(x)$_1$	6	5	5	4	4	4	4	3	1
DSAT(x)$_2$	■	1	1	1	4	1	1	1	1
DSAT(x)$_3$	■	2	2	2	■	2	1	1	1
DSAT(x)$_4$	■	■	3	3	■	3	1	1	1
DSAT(x)$_5$	■	■	■	4	■	3	1	2	1
DSAT(x)$_6$	■	■	■	■	■	3	1	2	1
DSAT(x)$_7$	■	■	■	■	■	■	2	2	1
DSAT(x)$_8$	■	■	■	■	■	■	■	3	1
DSAT(x)$_9$	■	■	■	■	■	■	■		
COULEUR	1	2	3	4	1	3	2		

Viens ensuite le sommet a qui a g, b et e comme sommet adjacents ; on le colorie avec la couleur 4.

Som($\mathcal{G}$)	g	b	e	a	d	f	h	c	i
DSAT(x)$_1$	6	5	5	4	4	4	4	3	1
DSAT(x)$_2$	■	1	1	1	4	1	1	1	1
DSAT(x)$_3$	■	2	2	2	■	2	1	1	1
DSAT(x)$_4$	■	■	3	3	■	3	1	1	1
DSAT(x)$_5$	■	■	■	4	■	3	1	2	1
DSAT(x)$_6$	■	■	■	■	■	3	1	2	1
DSAT(x)$_7$	■	■	■	■	■				
DSAT(x)$_8$	■	■	■	■	■				
DSAT(x)$_9$	■	■	■	■	■				
COULEUR	1	2	3	4	1				

Puisque le sommet c est adjacent aux sommet g, h et e on le colorie avec la couleur 4.

Som($\mathcal{G}$)	g	b	e	a	d	f	h	c	i
DSAT(x)$_1$	6	5	5	4	4	4	4	3	1
DSAT(x)$_2$	■	1	1	1	4	1	1	1	1
DSAT(x)$_3$	■	2	2	2	■	2	1	1	1
DSAT(x)$_4$	■	■	3	3	■	3	1	1	1
DSAT(x)$_5$	■	■	■	4	■	3	1	2	1
DSAT(x)$_6$	■	■	■	■	■	3	1	2	1
DSAT(x)$_7$	■	■	■	■	■	■	2	2	1
DSAT(x)$_8$	■	■	■	■	■	■	■	3	1
DSAT(x)$_9$	■	■	■	■	■	■	■	■	1
COULEUR	1	2	3	4	1	3	2	4	

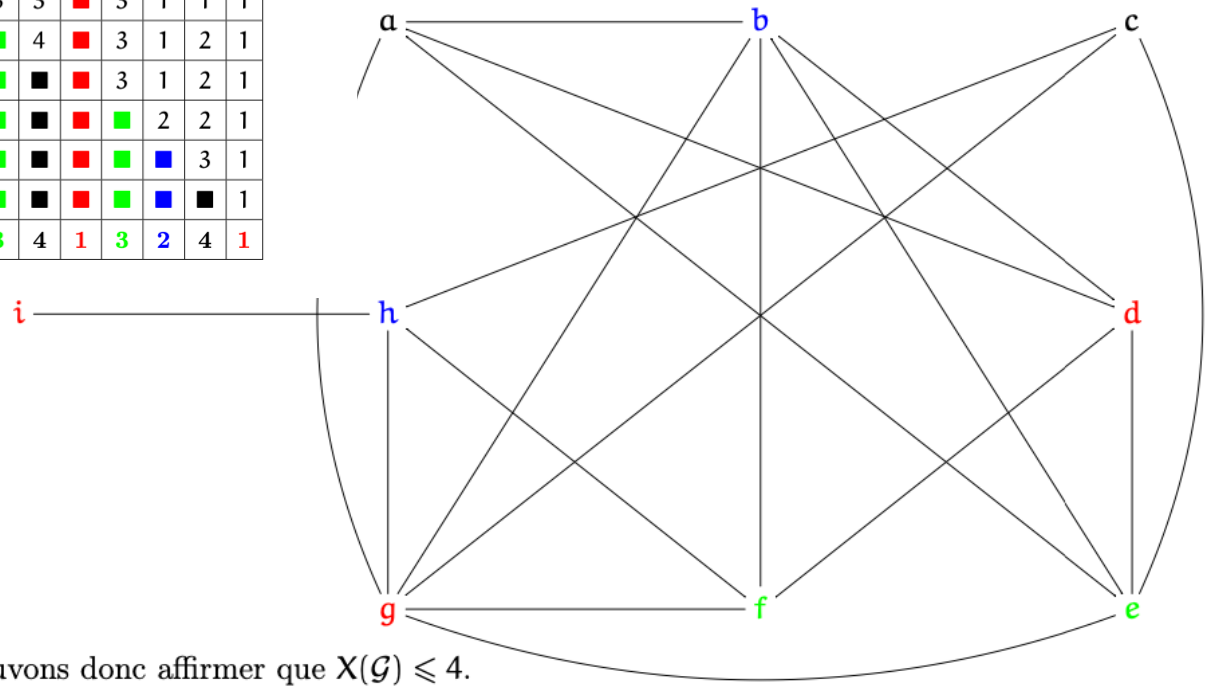
DSATUR algorithm : example, end

Le sommet f est adjacent à g et à b mais pas à e ; on le colorie donc avec la couleur 3.

Som($\mathcal{G}$)	g	b	e	a	d	f	h	c	i
DSAT(x)$_1$	6	5	5	4	4	4	4	3	1
DSAT(x)$_2$	■	1	1	1	4	1	1	1	1
DSAT(x)$_3$	■	2	2	2	■	2	1	1	1
DSAT(x)$_4$	■	■	3	3	■	3	1	1	1
DSAT(x)$_5$	■	■	■	4	■	3	1	2	1
DSAT(x)$_6$	■	■	■	■	■	3	1	2	1
DSAT(x)$_7$	■	■	■	■	■	■	2	2	1
DSAT(x)$_8$	■	■	■	■	■	■			
DSAT(x)$_9$	■	■	■	■	■	■			
COULEUR	1	2	3	4	1	3			

Et pour finir la couleur 1 convient au dernier sommet non colorié : i.

Som($\mathcal{G}$)	g	b	e	a	d	f	h	c	i
DSAT(x)$_1$	6	5	5	4	4	4	4	3	1
DSAT(x)$_2$	■	1	1	1	4	1	1	1	1
DSAT(x)$_3$	■	2	2	2	■	2	1	1	1
DSAT(x)$_4$	■	■	3	3	■	3	1	1	1
DSAT(x)$_5$	■	■	■	4	■	3	1	2	1
DSAT(x)$_6$	■	■	■	■	■	3	1	2	1
DSAT(x)$_7$	■	■	■	■	■	■	2	2	1
DSAT(x)$_8$	■	■	■	■	■	■	■	3	1
DSAT(x)$_9$	■	■	■	■	■	■	■	■	1
COULEUR	1	2	3	4	1	3	2	4	1



Nous pouvons donc affirmer que $X(\mathcal{G}) \leqslant 4$.

An application of graph coloring in scheduling

International Journal of Computational and Applied Mathematics.
ISSN 1819-4966 Volume 12, Number 2 (2017), pp. 469-485

http://www.ripublication.com

A Study on Course Timetable Scheduling using Graph Coloring Approach

Runa Ganguli[1] and Siddhartha Roy[2]

[1]*Department of Computer Science,*
The Bhawanipur Education Society College, Kolkata,West Bengal, India

[2] *Department of Computer Science,*
The Bhawanipur Education Society College, Kolkata,West Bengal, India

Abstract

In any educational institution, the two most common academic scheduling problems are course timetabling and exam timetabling. A schedule is desirable which combines resources like teachers, subjects, students, classrooms in a way to avoid conflicts satisfying various essential and preferential constraints. The timetable scheduling problem is known to be NP Complete but the corresponding optimization problem is NP Hard. Hence a heuristic approach is preferred to find a nearest optimal solution within reasonable running time. Graph coloring is one such heuristic algorithm that can deal timetable scheduling satisfying changing requirements, evolving subject demands and their combinations. This study shows application of graph coloring on

An application of graph coloring in scheduling

Input Dataset: Table 1 shows the Honours-General subject combination of a typical undergraduate science course.

Table 1 Honours-General Subject Combination

Sl. No.	List of Honours Subjects	General Subject Combination
1	Physics	Mathematics(compulsory) + Computer Science/Chemistry/Electronics
2	Chemistry	Mathematics(compulsory) + Physics/Computer Science
3	Mathematics	Physics(compulsory) + Chemistry/Computer Science/Statistics
4	Economics	Mathematics(compulsory) + Statistics/Computer Science

List of constraints:

Hard Constraints-

- Courses having common student cannot be allotted at the same time slot on the same day.
- Total number of available periods is 8. (maximum)

Soft Constraints-

- Honours and General courses need to be scheduled in non-overlapping time-slots.

Solution:

Considering each course as a node, edge between two nodes is drawn only if there is common student. (See Fig. 3)

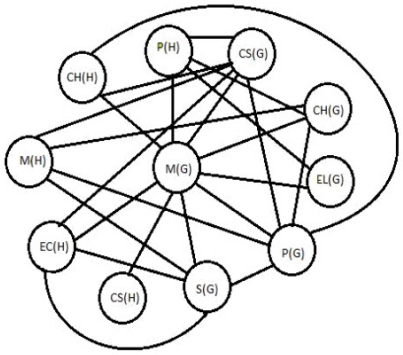


Fig. 3 Course Conflict Graph

After applying graph coloring algorithm, the resultant graph in Fig. 4 is properly colored with chromatic number 4. This is the minimum number of non-conflicting time-slots scheduling all the given courses.

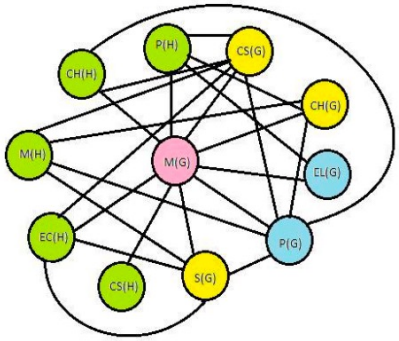


Fig. 4 Colored Conflict Graph

Exercise: application in scheduling

Suppose that in a particular quarter there are students taking each of the following combinations of courses:

- *Math, English, Biology, Chemistry*
- *Math, English, Computer Science, Geography*
- *Biology, Psychology, Geography*, Spanish
- *Biology, Computer Science, History, French*
- *English, Psychology, Computer Science, History*
- *Psychology, Chemistry, Computer Science, French*
- *Psychology, Geography*, *History, Spanish*

What is the minimum number of examination periods required for the exams in the ten courses specified so that students taking any of the given combinations of courses have no conflicts? Find a schedule that uses this minimum number of periods.

Exercise: application in scheduling (2)

Suppose that in a particular quarter there are students taking each of the following combinations of courses:

- *Math, English, Biology, Chemistry*
- *Math, English, Computer Science, Geography*
- *Biology, Psychology, Geography*, Spanish
- *Biology, Computer Science, History, French*
- *English, Psychology, Computer Science, History*
- *Psychology, Chemistry, Computer Science, French*
- *Psychology, Geography, History, Spanish*

Create a Conflict Graph

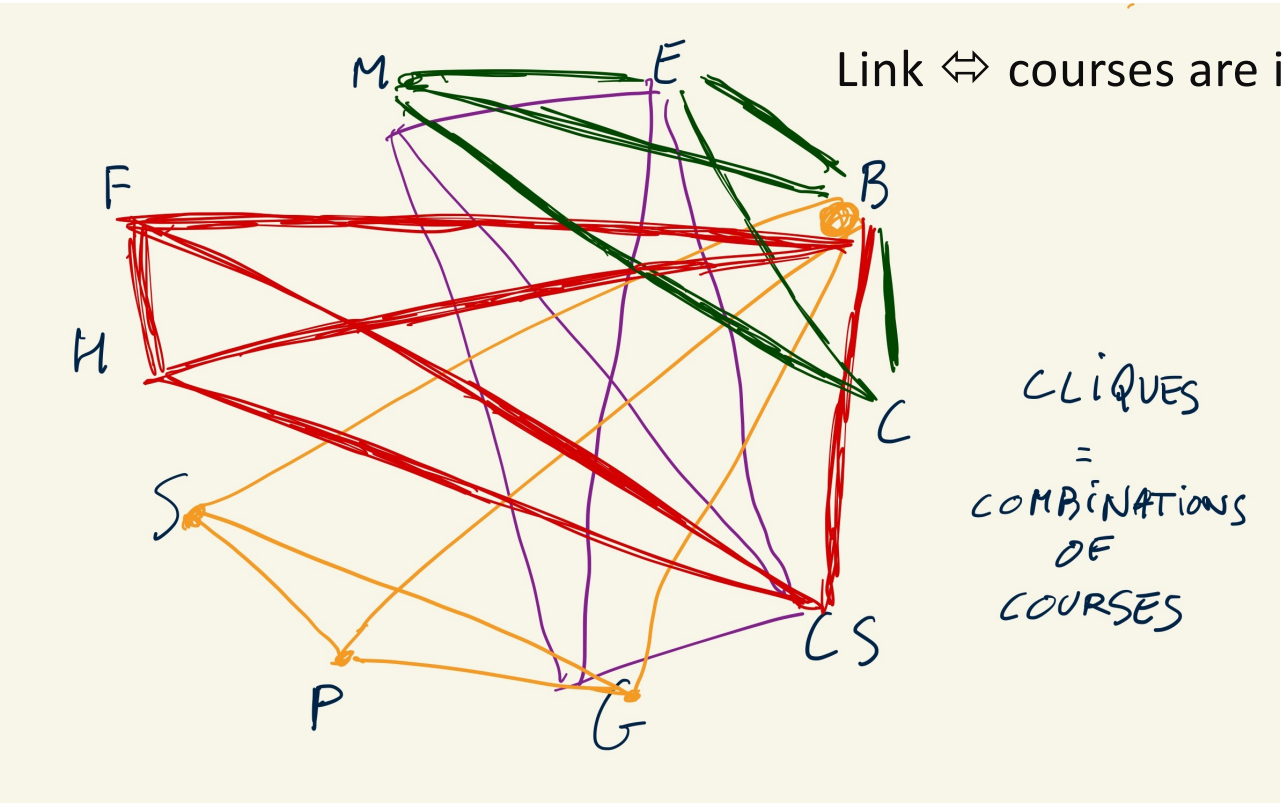
• Each node represents a course.

• An edge between two courses signifies that they have a common student and thus cannot be in the same exam period.

Math (M), English (E), Biology (B), Chemistry (C), Computer Science (CS), Geography (G), Psychology (P), Spanish (S), History (H), and French (F).

Exercise: application in scheduling (2)

Math (M)
English (E)
Biology (B)
Chemistry (C)
Computer Science (CS)
Geography (G)
Psychology (P)
Spanish (S)
History (H)
French (F)



From the combinations of courses given, we can infer the edges.
For example, from the first combination (Math, English, Biology, Chemistry),
we draw edges between M and E, N and B, M and C, E and B, E and C, and B and C.

Exercise: application in scheduling (3)

With the graph in place, we apply a graph coloring algorithm to determine the minimum number of colors (examination periods) needed.

The chromatic number of a graph is the smallest number of colors needed to color the vertices of the graph so that no two adjacent vertices share the same color.

Finding the chromatic number of a general graph is NP-Complete, but for small graphs, we can often find it by inspection or using a straightforward **greedy algorithm**.