

Vertex coloring: the DSATUR branch-and-bound



Outline

The algorithm for the VCP

- Branching

- Bounds and pruning

- Node selection and pseudocode

- Correctness

A worked example

Remarks

Typical mistakes

In summary

Learning objectives

At the end of this chapter you will be able to:

- ▶ describe a node of the tree as a partial coloring, with its colors used and its saturation degrees;
- ▶ branch on the vertex of maximum saturation degree, with one child per feasible color and one new color;
- ▶ compute the upper bound with *DSATUR*, the lower bound with a greedy clique, and prune a node;
- ▶ execute the *DSATUR* branch-and-bound algorithm by hand and prove that it is exact.

Prerequisites

- ▶ From the module *Integer linear programming*, the chapter «[Branch-and-bound](#)»: branching and bounding operations, the problem list, the pruning criteria, the bounds LB and UB.
- ▶ From this module, the chapter «[Coloring problems](#)»: the vertex coloring problem, cliques, stable sets and the clique bound $\omega(G) \leq \chi(G)$.
- ▶ From this module, the chapter «[Constructive heuristics](#)»: the saturation degree and the *DSATUR* heuristic.

The problem of this chapter

The problem of this chapter is the
vertex coloring problem (VCP):

its definition and its assumptions are on its page.

- ▶ On a graph $G = (V, E)$, its optimal value is the chromatic number $\chi(G)$
- ▶ We write an exact algorithm for the VCP: a branch-and-bound algorithm whose nodes are partial colorings, and whose bounds come from counting colors

The algorithm for the VCP

Introduction: The algorithm for the VCP

- ▶ In this section we build the DSATUR branch-and-bound algorithm, piece by piece: the nodes of the tree, the branching rule, the bounds and the pruning, the node selection
- ▶ The branching rule is the rule of the *DSATUR* heuristic: color first the most constrained vertex
- ▶ At the end we prove that the algorithm is exact, and that its tree contains no symmetric copies of the same coloring

The nodes of the tree

- ▶ In a branch-and-bound algorithm for the VCP, a node of the tree must describe a set of colorings
- ▶ Here a node is a **partial coloring**: some vertices already have a color, the others do not
- ▶ We first define a partial coloring and the quantities we compute at a node

Branching node and partial coloring (1/2)

Definition 1: branching node and partial coloring

*In the DSATUR branch-and-bound algorithm, the colors are the integers $1, 2, 3, \dots$, and a branching node k of the tree is a **partial coloring**:*

1. *a subset of **colored vertices** $\rightsquigarrow$ denoted $V^k \subseteq V$;*
2. *a color $col^k(v)$ for every $v \in V^k$, with $col^k(u) \neq col^k(v)$ for all $\{u, v\} \in E$, $u, v \in V^k$;*
3. *a subset of **uncolored vertices** $\rightsquigarrow$ denoted $U^k := V \setminus V^k$.*

*The colors used at node k are $1, 2, \dots, q(k)$, where $q(k)$ is the **number of colors used**, and the color classes are $Q_c^k := \{v \in V^k : col^k(v) = c\}$, $c = 1, \dots, q(k)$.*

Branching node and partial coloring (2/2)

Definition: branching node and partial coloring (continued)

For an uncolored vertex $v \in U^k$:

- ▶ the *saturation degree*

$$\text{sat}_k(v) := |\{ \text{col}^k(u) : u \in N(v) \cap V^k \}|;$$

- ▶ the *degree among the uncolored vertices*

$$\text{du}_k(v) := |N(v) \cap U^k|;$$

- ▶ the *feasible colors*

$$F_k(v) := \{ c \in \{1, \dots, q(k)\} : Q_c^k \cap N(v) = \emptyset \}.$$

Saturation degree and complete colorings

- ▶ The saturation degree counts the *different* colors among the colored neighbours of v , and

$$|F_k(v)| = q(k) - \text{sat}_k(v).$$

- ▶ A node with $U^k = \emptyset$ is a **complete coloring**: a feasible solution of the VCP with $q(k)$ colors.

Nodes as partial colorings

A node of the tree is a partial coloring; each child colors one more vertex.

No LP problem is solved: the bounds of a node come from counting colors.

Branching

Now that we know what a node is,
we decide which vertex to color next, and with which colors.

The branching vertex

- ▶ At node k we branch on the uncolored vertex v of **maximum saturation degree** $sat_k(v)$
- ▶ Ties are broken by the largest $du_k(v)$, and then by the smallest index
- ▶ This is the rule of the *DSATUR* heuristic (Brélaz, 1979)

The children of a node

The children of node k give a color to v :

1. one child for every feasible color $c \in F_k(v)$, in increasing order of c ;
 2. one child with the **new color** $q(k) + 1$.
- ▶ The colors $q(k) + 2, q(k) + 3, \dots$ are not used: a child with color $q(k) + 2$ would differ from the child with color $q(k) + 1$ only by the name of an empty color
 - ▶ The children are **no symmetric copies** of each other
 - ▶ The most saturated vertex has the fewest children: $|F_k(v)| + 1$.
If $F_k(v) = \emptyset$, the node has only the child with the new color

The branching rule

The branching vertex is the most constrained one.

Its children are its feasible colors plus one new color, so two children never describe the same coloring with the colors renamed.

Bounds and pruning

After the branching, the bounds:
an upper bound from a heuristic, a lower bound from a clique,
and a pruning rule that only counts colors.

Upper bound and greedy clique

- ▶ The VCP is a minimization problem: the incumbent solution gives the **upper bound** UB, and a lower bound comes from a clique
- ▶ **Upper bound.** Before branching, the *DSATUR* heuristic of the chapter «[Constructive heuristics](#)» gives the first incumbent solution, and UB is its number of colors
- ▶ **Greedy clique.** The vertices are scanned by non-increasing degree (ties: smallest index), and a vertex enters K if it is adjacent to all the vertices already in K . The set $K = \{k_1, \dots, k_r\}$ is a maximal clique, and

$$\text{LB} := r \leq \omega(G) \leq \chi(G).$$

Pre-coloring and lower bound of a node

- ▶ **Pre-coloring.** The vertices of K need r different colors in every coloring, and the names of the colors are arbitrary: the root node colors k_i with color i , $i = 1, \dots, r$. This breaks the symmetry of the first r colors
- ▶ **Lower bound of a node.** The colors of a node are never removed in its subtree, so every complete coloring below node k uses at least $q(k)$ colors:

$$\text{LB}(k) := \max\{q(k), |K|\} = q(k),$$

where the equality holds because the clique is pre-colored

Pruning and complete colorings

- ▶ **Pruning.** The number of colors is an integer, so a node can lead to a better solution only if it uses fewer than UB colors. Node k is **pruned by bound** if

$$q(k) \geq \text{UB}.$$

- ▶ **Complete coloring.** A node with $U^k = \emptyset$ and $q(k) < \text{UB}$ is a new incumbent solution: $\text{UB} \leftarrow q(k)$, and the nodes of the list with $q \geq \text{UB}$ are removed
- ▶ If $\text{UB} = |K|$, the incumbent solution is optimal and the algorithm stops

The bounds of the algorithm

DSATUR gives UB, a greedy clique gives LB.

A node is pruned as soon as it uses UB colors, because its subtree cannot contain a coloring with fewer colors.

Node selection

- ▶ The tree is explored **depth first**: the list L is a stack, the children of a node are evaluated in increasing order of color, and the child with the smallest color is extracted first
- ▶ The nodes are numbered in the order of evaluation
- ▶ A child is evaluated as soon as it is created: the evaluation only computes q and checks whether the coloring is complete
- ▶ We now put all the pieces together in the pseudocode of the algorithm

The DSATUR branch-and-bound algorithm (1/2)

Input: a VCP instance, i.e., a graph $G = (V, E)$.

Output: an optimal coloring and the chromatic number $\chi(G)$.

Step 0: Initialization

- (a) Run *DSATUR*: set the incumbent solution to its coloring and UB to its number of colors.
- (b) Find the greedy clique $K = \{k_1, \dots, k_r\}$ and set $LB \leftarrow r$. If $LB = UB$, the incumbent solution is optimal. STOP.
- (c) Create node 1 with $col^1(k_i) = i$, $i = 1, \dots, r$, and $q(1) = r$. Insert node 1 into the list L and set the node counter $c \leftarrow 1$.

The DSATUR branch-and-bound algorithm (2/2)

Step 1: Optimality check

If L is empty, UB is the optimal value and the incumbent solution is optimal. STOP.

Step 2: Branching-node selection

Remove from L the node k inserted last.

Step 3: Branching

Select $v \in U^k$ of maximum $sat_k(v)$, then maximum $du_k(v)$, then smallest index. For each color $a \in F_k(v) \cup \{q(k) + 1\}$, in increasing order, create node $c + 1$ as node k with $col(v) = a$, evaluate it with the **node sub-procedure**, and set $c \leftarrow c + 1$. Insert the surviving children into L in decreasing order of color and go to Step 1.

The node sub-procedure

- ▶ In Step 3, the counter c of the nodes gives the next free index, and a is the color given to v in the child
- ▶ Each child is evaluated by the **node sub-procedure**: it applies the bounding operations and decides whether the node survives
- ▶ Its pseudocode is on the next slide

The node sub-procedure: pseudocode

Input: a node h , i.e., a partial coloring with $q(h)$ colors.

Step 1: Bounding operations

- (a) If $q(h) \geq \text{UB}$, the node is pruned by bound. STOP.
- (b) If $U^h = \emptyset$, update the upper bound and the incumbent solution: $\text{UB} \leftarrow q(h)$. Remove from L all the nodes with $q \geq \text{UB}$. If $\text{UB} = \text{LB}$, the incumbent solution is optimal: STOP the algorithm. Otherwise STOP.

Step 2: Survival

The node survives: it is inserted into L in Step 3.
STOP.

Is the algorithm exact?

We first look at the tree without pruning:
which colorings does it contain, and how many times?

No-symmetry branching

Proposition 1: no-symmetry branching

*Consider the tree of the DSATUR branch-and-bound algorithm without pruning. Its leaves are complete colorings, and every partition of V into stable sets, i.e., every feasible solution of the VCP, is the set of color classes of **exactly one** leaf.*

- ▶ The proof has two parts: every partition is reached (**existence**), and it is reached only once (**uniqueness**)

Proof (1/3)

- ▶ Let $\mathcal{Q}$ be a partition of V into stable sets
- ▶ The vertices of the clique K are pairwise adjacent, so they lie in r different classes; we give the name i to the class of k_i
- ▶ **Existence.** We follow a path from the root and keep this invariant: the classes of the node are the classes of $\mathcal{Q}$ restricted to V^k , each with its name
- ▶ The root satisfies it. At node k , let v be the branching vertex and $Q \in \mathcal{Q}$ its class

Proof (2/3)

Existence (continued). Two cases:

- ▶ If Q already has a colored vertex, with name c , then no neighbour of v has color c , because Q is stable. Hence $c \in F_k(v)$, and the child with color c keeps the invariant
- ▶ Otherwise Q has no colored vertex yet: the child with the new color $q(k) + 1$ keeps the invariant, and Q gets the name $q(k) + 1$

Every step colors one vertex, so after $|U^1|$ steps we reach a leaf whose classes are $\mathcal{Q}$.

Proof (3/3)

Uniqueness.

- ▶ Two different children of a node put v in different classes: with the colored vertices of two different colors, or with one of them and alone
- ▶ The partition restricted to $V^k \cup \{v\}$ differs, so the leaves below two different children give different partitions



The algorithm is exact

- ▶ **The algorithm is exact.** A node is pruned only if $q(k) \geq \text{UB}$, and every leaf below it uses at least $q(k)$ colors: no coloring with fewer than UB colors is lost. When the list is empty, $\text{UB} = \chi(G)$
- ▶ Without the new-color rule, every partition with q classes would appear once for every way of naming its classes with the colors $1, \dots, \text{UB} - 1$: the tree would contain many symmetric copies of the same solution

Enumeration without symmetry

Without pruning, the tree enumerates every feasible solution of the VCP exactly once, up to a renaming of the colors.

The pruning by bound removes only subtrees with at least UB colors: the algorithm is exact.

Summary (1/2)

Nodes:

- ▶ a node is a partial coloring: colored vertices V^k , uncolored vertices U^k , $q(k)$ colors used
- ▶ for $v \in U^k$: saturation degree $sat_k(v)$, degree $du_k(v)$, feasible colors $F_k(v)$

Branching:

- ▶ on the uncolored vertex of maximum $sat_k(v)$, then maximum $du_k(v)$, then smallest index
- ▶ one child per feasible color, plus one child with the new color $q(k) + 1$

Summary (2/2)

Bounds and pruning:

- ▶ UB from *DSATUR*; $LB = |K|$ from a greedy clique, pre-colored at the root
- ▶ a node is pruned by bound when $q(k) \geq UB$

Search and correctness:

- ▶ depth-first search: the list L is a stack
- ▶ without pruning, every feasible coloring is a leaf exactly once: the algorithm is exact

A worked example

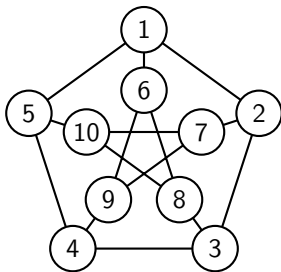
Introduction: A worked example

- ▶ In this section we run the DSATUR branch-and-bound algorithm by hand on the Petersen graph
- ▶ We follow the rules of the previous section: branching on the most saturated vertex, depth-first search, children in increasing order of color
- ▶ The tree is small, but it proves that the coloring found by *DSATUR* is optimal

Example: the Petersen graph

We consider the Petersen graph, the reference instance of the chapter «[Coloring problems](#)», with $n = 10$ vertices, $m = 15$ edges and all degrees equal to 3:

$$E = \{ \{1, 2\}, \{1, 5\}, \{1, 6\}, \{2, 3\}, \{2, 7\}, \{3, 4\}, \{3, 8\}, \{4, 5\}, \\ \{4, 9\}, \{5, 10\}, \{6, 8\}, \{6, 9\}, \{7, 9\}, \{7, 10\}, \{8, 10\} \}.$$



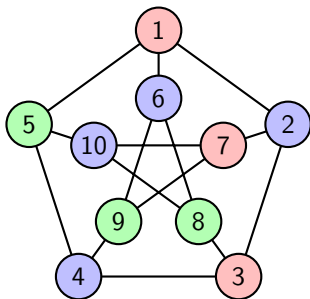
Example: initialization (1/3)

The *DSATUR* heuristic (example «*DSATUR* on the Petersen graph» of the chapter «[Constructive heuristics](#)»):

- ▶ colors the vertices in the order
1, 2, 3, 4, 5, 6, 8, 7, 9, 10
- ▶ gives the incumbent solution

$$\bar{Q} = \{ \{1, 3, 7\}, \{2, 4, 6, 10\}, \\ \{5, 8, 9\} \},$$

and $UB = 3$



Example: initialization (2/3)

- ▶ All the degrees are equal, so the greedy clique scans the vertices $1, 2, \dots, 10$
- ▶ Vertex 1 enters, vertex 2 is adjacent to 1 and enters, and no other vertex is adjacent to both
- ▶ Hence

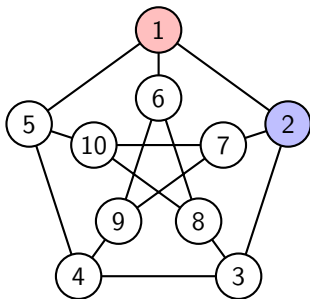
$$K = \{1, 2\} \quad \text{and} \quad \text{LB} = 2$$

- ▶ The Petersen graph has no triangle, so $\omega(G) = 2$

Example: initialization (3/3)

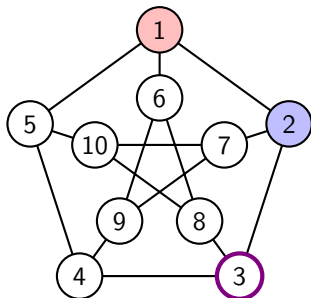
- ▶ Node 1 colors $col^1(1) = 1$ and $col^1(2) = 2$, with $q(1) = 2$
- ▶ Since $LB = 2 < 3 = UB$, node 1 is inserted into the list L :

$$2 \leq \chi(G) \leq 3.$$



Example: iteration 1

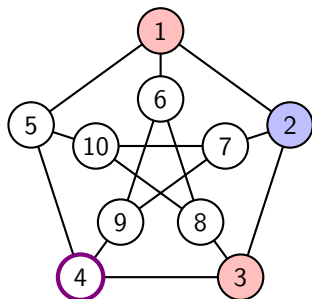
- ▶ Node 1 is extracted from L
- ▶ The vertices 3, 5, 6, 7 have one colored neighbour each, so $sat_1 = 1$; the others have $sat_1 = 0$
- ▶ The four tied vertices have $du_1 = 2$, so the smallest index wins: the branching vertex is 3
- ▶ Its neighbour 2 has color 2, so $F_1(3) = \{1\}$



1. Node 2: $col(3) = 1$, with $q(2) = 2 < UB$. The node survives.
2. Node 3: $col(3) = 3$, the new color, with $q(3) = 3 \geq UB$. The node is pruned by bound.

Example: iteration 2

- ▶ Node 2 is extracted from L , with $Q_1 = \{1, 3\}$ and $Q_2 = \{2\}$
- ▶ The vertices 4, 5, 6, 7, 8 have $sat_2 = 1$ and $du_2 = 2$, and 9, 10 have $sat_2 = 0$: the branching vertex is 4
- ▶ Its neighbour 3 has color 1, so $F_2(4) = \{2\}$



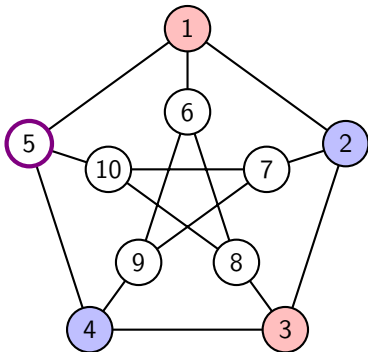
1. Node 4: $col(4) = 2$, with $q(4) = 2 < UB$. The node survives.
2. Node 5: $col(4) = 3$, with $q(5) = 3 \geq UB$. The node is pruned by bound.

Example: iteration 3 (1/2)

- ▶ Node 4 is extracted from L , with $Q_1 = \{1, 3\}$ and $Q_2 = \{2, 4\}$
 - ▶ Vertex 5 has the neighbours 1 (color 1) and 4 (color 2):
 $sat_4(5) = 2$, the only vertex with saturation degree 2, so it is the branching vertex
 - ▶ Its feasible colors are $F_4(5) = \emptyset$
1. Node 6: $col(5) = 3$, the only child, with $q(6) = 3 \geq UB$. The node is pruned by bound.

Example: iteration 3 (2/2)

The partial coloring of node 4 is drawn below: the cycle 1, 2, 3, 4, 5 has five vertices, and two colors are not enough to close it.



Example: termination

- ▶ The list L is empty: the incumbent solution $\bar{Q}$ is optimal and

$$\chi(G) = \text{UB} = 3$$

- ▶ The tree has 6 nodes

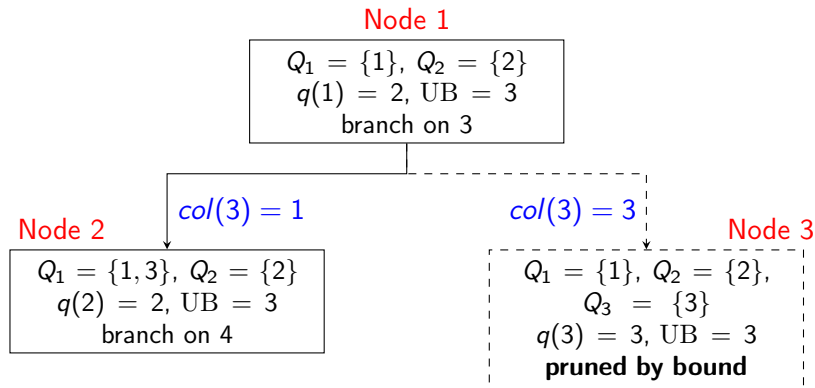
Next, we draw the branch-and-bound tree, step by step.

The branch-and-bound tree

- ▶ For each node, the tree shows:
 - ▶ its color classes
 - ▶ the number of colors q
 - ▶ the upper bound UB at the moment of the evaluation
 - ▶ the branching vertex or the pruning criterion
- ▶ The pruned nodes are dashed
- ▶ The tree is too large for one slide: we draw it one iteration at a time, and then as a whole in a compact form

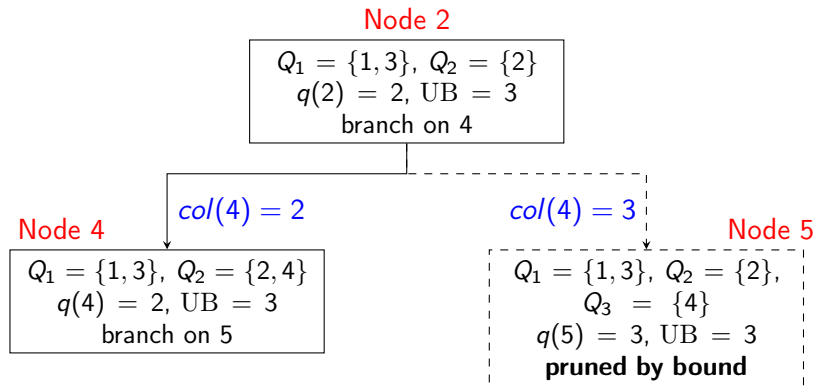
The branch-and-bound tree (1/4)

Iteration 1: node 1 and its children.



The branch-and-bound tree (2/4)

Iteration 2: node 2 and its children.



The branch-and-bound tree (3/4)

Iteration 3: node 4 and its only child.

Node 4

$Q_1 = \{1, 3\}, Q_2 = \{2, 4\}$
 $q(4) = 2, \text{UB} = 3$
branch on 5

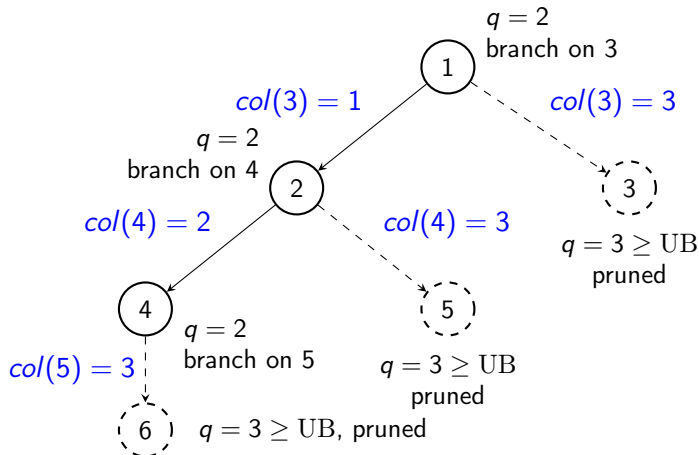
$col(5) = 3$

Node 6

$Q_1 = \{1, 3\}, Q_2 = \{2, 4\},$
 $Q_3 = \{5\}$
 $q(6) = 3, \text{UB} = 3$
pruned by bound

The branch-and-bound tree (4/4)

The whole tree, in compact form: the pruned nodes are dashed.



The nodes in the order of evaluation

Node	Parent	Child	q	UB	Branching vertex	Outcome
1	–	root, $K = \{1, 2\}$	2	3	3 ($sat = 1, du = 2$)	branched
2	1	$col(3) = 1$	2	3	4 ($sat = 1, du = 2$)	branched
3	1	$col(3) = 3$	3	3	–	pruned by bound
4	2	$col(4) = 2$	2	3	5 ($sat = 2, du = 1$)	branched
5	2	$col(4) = 3$	3	3	–	pruned by bound
6	4	$col(5) = 3$	3	3	–	pruned by bound

What the example shows

- ▶ The algorithm does not improve the incumbent solution: it proves that $UB = 3$ is optimal
- ▶ The proof is the odd cycle 1, 2, 3, 4, 5, found by the search in three levels
- ▶ Without the pruning, the tree of the Petersen graph has 9 776 nodes and 7 430 leaves, one for each feasible partition counted in the chapter «[Coloring problems](#)», as the no-symmetry proposition predicts

Summary

Initialization:

- ▶ *DSATUR* gives $UB = 3$; the greedy clique $K = \{1, 2\}$ gives $LB = 2$
- ▶ node 1 pre-colors the clique: $2 \leq \chi(G) \leq 3$

Search:

- ▶ three branchings, on the vertices 3, 4 and 5; every child with $q = 3$ is pruned by bound
- ▶ the list is empty after 6 nodes: $\chi(G) = 3$, proved by the odd cycle 1, 2, 3, 4, 5

Remarks

Introduction: Remarks

- ▶ In this section we see where the algorithm comes from
- ▶ We also see two refinements of its branching rule, which change only how ties are broken
- ▶ The references are at the end of the section

Origin and Sewell's tie-breaking

- ▶ **Origin.** The saturation degree and the exact algorithm that branches on it are due to Brélaz (1979). Pre-coloring a clique is the usual way to remove the symmetry of the first colors
- ▶ **Sewell's tie-breaking (1996).** Among the vertices of maximum saturation degree, Sewell chooses the vertex v that maximizes the number of feasible colors it shares with its uncolored neighbours,

$$\sum_{u \in N(v) \cap U^k} |F_k(v) \cap F_k(u)|.$$

Coloring that vertex removes the most options from the rest of the graph

- ▶ **PASS (San Segundo, 2012).** PASS applies Sewell's rule only to selected vertices, computing the count over the vertices tied for the maximum saturation degree, which is cheaper
- ▶ Both rules change only the tie-breaking: the children, the bounds and the no-symmetry proposition stay the same

References

- ▶ D. Brélaz, «New methods to color the vertices of a graph», *Communications of the ACM* 22(4), 251–256, 1979
- ▶ E. C. Sewell, «An improved algorithm for exact graph coloring», in D. S. Johnson, M. A. Trick (eds.), *Cliques, Coloring, and Satisfiability*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 26, 1996
- ▶ P. San Segundo, «A new DSATUR-based algorithm for exact vertex coloring», *Computers & Operations Research* 39(7), 1724–1733, 2012

Summary

Origin:

- ▶ saturation degree and exact algorithm: Brélaz (1979)
- ▶ pre-coloring a clique removes the symmetry of the first colors

Tie-breaking:

- ▶ Sewell (1996): the vertex sharing the most feasible colors with its uncolored neighbours
- ▶ PASS (San Segundo, 2012): Sewell's count only on the tied vertices
- ▶ children, bounds and the no-symmetry proposition do not change

Typical mistakes

Introduction: Typical mistakes

- ▶ We collect the mistakes that are made most often with the DSATUR branch-and-bound algorithm
- ▶ Two concern the saturation degree and the children, two the pruning, two the clique
- ▶ Check them before running the algorithm by hand

Typical mistakes (1/3)

Observation 1: Counting colored neighbours instead of colors

The saturation degree counts the different colors around v : two neighbours with the same color count once.

Observation 2: One child per color $1, \dots, \text{UB} - 1$

The unused colors are all equivalent: only the first new color $q(k) + 1$ gives a child, otherwise the tree contains symmetric copies of the same coloring.

Typical mistakes (2/3)

Observation 3: Forgetting the new color

The child with color $q(k) + 1$ is always created, even when $F_k(v) \neq \emptyset$; without it some colorings are never reached.

Observation 4: Pruning with $q(k) > \text{UB}$

A node with $q(k) = \text{UB}$ cannot lead to a better coloring, because the number of colors is an integer: the rule is $q(k) \geq \text{UB}$.

Typical mistakes (3/3)

Observation 5: Pre-coloring a set that is not a clique

Only the vertices of a clique need pairwise different colors in every coloring; pre-coloring two non-adjacent vertices with different colors can cut off all the optimal colorings.

Observation 6: Continuing after $UB = |K|$

When a complete coloring reaches the clique bound, it is optimal: the nodes still in the list are not explored.

Summary

Saturation degree and children:

- ▶ the saturation degree counts different colors, not colored neighbours
- ▶ the children are the feasible colors plus the new color $q(k) + 1$, always and only that one

Pruning and clique:

- ▶ prune when $q(k) \geq \text{UB}$, not only when $q(k) > \text{UB}$
- ▶ pre-color only a clique, and stop as soon as $\text{UB} = |K|$

In summary

Introduction: In summary

- ▶ We collect the key ideas of the chapter on the DSATUR branch-and-bound algorithm
- ▶ The dashboard collects the bounds on $\chi(G)$ found so far for the Petersen graph

Summary (1/2)

- ▶ A node of the DSATUR branch-and-bound is a partial coloring; the algorithm branches on the most saturated vertex, with one child per feasible color and one new color.
- ▶ *DSATUR* gives the first upper bound, a greedy clique gives the lower bound and is pre-colored; a node is pruned when $q(k) \geq \text{UB}$.
- ▶ Without pruning the tree enumerates every feasible coloring once, up to a renaming of the colors: the algorithm is exact.

Summary (2/2)

Dashboard of the Petersen graph ($n = 10$, $m = 15$). The bounds on $\chi(G)$ found so far:

Source	Lower bound	Upper bound	From
Clique bound («Coloring problems»)	2	–	$\omega(G) = 2$
<i>DSATUR</i> («Constructive heuristics»)	–	3	$\{1, 3, 7\}$, $\{2, 4, 6, 10\}$, $\{5, 8, 9\}$
This chapter, initialization	2	3	greedy clique $K = \{1, 2\}$
This chapter, termination	3	3	empty list, 6 nodes