



SAPIENZA
UNIVERSITÀ DI ROMA

Combinatorial Optimization

Volume 6 — Ad hoc algorithms

Ad hoc algorithms: the branch-and-bound on the critical item for the 0/1 knapsack problem and the DSATUR branch-and-bound for vertex coloring.

Fabio Furini

fabiofurini.github.io

Department of Computer, Control and Management Engineering — Sapienza University of Rome

Online version and downloadable material:
fabiofurini.github.io/combinatorial-optimization/
github.com/fabiofurini/combinatorial-optimization

Academic year 2026–2027

Contents

Knapsack problems	2
41 Knapsack: branch-and-bound on the critical item	3
41.1 The algorithm for the KP	4
41.2 A worked example	6
41.3 Typical mistakes	12
41.4 In summary	12
 Coloring problems	 13
42 Vertex coloring: the DSATUR branch-and-bound	14
42.1 The algorithm for the VCP	15
42.2 A worked example	19
42.3 Remarks	21
42.4 Typical mistakes	22
42.5 In summary	22

Knapsack problems



Knapsack: branch-and-bound on the critical item

Prerequisites

- From the module *Integer linear programming*, [Chapter 7 «Branch-and-bound»](#): branching and bounding operations, the problem list, the pruning criteria, the lower bound LB and the upper bound UB. The general algorithm is applied here to the 0/1 knapsack problem.
- From this module, [Chapter 9 «Formulations of knapsack problems»](#): the models ILP_{KP} and LP_{KP} , the efficiency of an item, the split item s , the residual capacity rc and the bound of Dantzig.

Objectives

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

- write the problem LP_{KP}^k of a node, and solve it with the formulas of Dantzig, without the simplex algorithm;
- branch on the variable of the split item, unselecting or selecting that item;
- execute the branch-and-bound algorithm for the KP by hand, with the FIFO or the LIFO rule;
- explain why the algorithm is simpler than the general one, and what stays the same.

The problem of this chapter is the [0/1 knapsack problem \(KP\)](#): its definition and its assumptions are on its page.

41.1 The algorithm for the KP

Definition 41.1: branching node and LP_{KP}^k problem

In the branch-and-bound algorithm for the ILP_{KP} model, the branching constraints on a variable x_j , $j \in \mathcal{J}$, are $x_j \leq 0$ and $x_j \geq 1$. In a branching node k of the branch-and-bound tree, we have:

1. a subset of **unselected items** $\rightsquigarrow$ denoted $\mathcal{J}_{out}^k$
2. a subset of **selected items** $\rightsquigarrow$ denoted $\mathcal{J}_{in}^k$
3. a subset of **remaining items** $\rightsquigarrow$ denoted $\mathcal{J}^k := \mathcal{J} \setminus \{\mathcal{J}_{out}^k \cup \mathcal{J}_{in}^k\}$

The LP_{KP} problem associated to node k , called the LP_{KP}^k problem, reads as follows:

$$\begin{aligned}
 & \max \sum_{j \in \mathcal{J}} p_j x_j \\
 & \text{subject to} \quad \sum_{j \in \mathcal{J}} w_j x_j \leq c, \\
 & \qquad \qquad \qquad x_j \geq 1, \qquad \qquad \qquad \forall j \in \mathcal{J}_{in}^k, \\
 & \qquad \qquad \qquad x_j \leq 0, \qquad \qquad \qquad \forall j \in \mathcal{J}_{out}^k, \\
 & \qquad \qquad \qquad x_j \leq 1, \qquad \qquad \qquad \forall j \in \mathcal{J}, \\
 & \qquad \qquad \qquad x_j \geq 0, \qquad \qquad \qquad \forall j \in \mathcal{J}.
 \end{aligned}$$

- The branching constraints $x_j \leq 0$ and $x_j \geq 1$ have the effect of **unselecting** or **selecting** an item.

A LP_{KP}^k problem of a branching node k can be equivalently re-written as:

$$\underbrace{\sum_{j \in \mathcal{J}_{in}^k} p_j}_{=p(k)} + \max \sum_{j \in \mathcal{J}^k} p_j x_j \tag{41.1a}$$

$$\text{subject to} \quad \sum_{j \in \mathcal{J}^k} w_j x_j \leq c - \underbrace{\sum_{j \in \mathcal{J}_{in}^k} w_j}_{=c(k)}, \tag{41.1b}$$

$$x_j \leq 1, \qquad \qquad \qquad \forall j \in \mathcal{J}^k, \tag{41.1c}$$

$$x_j \geq 0, \qquad \qquad \qquad \forall j \in \mathcal{J}^k. \tag{41.1d}$$

- A LP_{KP}^k problem of a branching node k is **structurally equivalent** to the LP_{KP} problem but:

1. the set of items is $\mathcal{J}^k$ instead of $\mathcal{J}$
2. the capacity is reduced to $c(k) := c - \sum_{j \in \mathcal{J}_{\text{in}}^k} w_j$
3. there is an objective-function constant $p(k) := \sum_{j \in \mathcal{J}_{\text{in}}^k} p_j$

41.1.1 Solving the LP_{KP}^k problem of a node

- The problem LP_{KP}^k is a knapsack LP problem on the items of $\mathcal{J}^k$, with capacity $c(k)$: it is solved by the formulas of Dantzig, without any simplex iteration. The items of $\mathcal{J}^k$ are kept in the order of non-increasing efficiency.
- If $c(k) < 0$ the problem is **infeasible**. If $\sum_{j \in \mathcal{J}^k} w_j \leq c(k)$ all the items of $\mathcal{J}^k$ fit, and the optimal solution is $\tilde{x}_j = 1$ for all $j \in \mathcal{J}^k$: it is integer.
- Otherwise the **split item** s of the node, also called the **critical item**, is the first item of $\mathcal{J}^k$ that does not fit entirely, and the **residual capacity** is the capacity left by the items that precede it:

$$\sum_{j \in \mathcal{J}^k, j < s} w_j \leq c(k) < \sum_{j \in \mathcal{J}^k, j \leq s} w_j, \quad rc = c(k) - \sum_{j \in \mathcal{J}^k, j < s} w_j.$$

- The optimal solution is $\tilde{x}_j = 1$ for the items of $\mathcal{J}^k$ before s , $\tilde{x}_s = \frac{rc}{w_s}$ and $\tilde{x}_j = 0$ for the items after s , and the optimal value is

$$z(\text{LP}_{\text{KP}}^k) = p(k) + \sum_{j \in \mathcal{J}^k, j < s} p_j + p_s \frac{rc}{w_s}.$$

In every node at most one variable is fractional, the one of the split item, and the value of the node is the Dantzig bound of a smaller knapsack: the node is solved by a scan of the items.

41.1.2 Branching-variable selection strategy

- The optimal solution of an LP_{KP}^k problem given by the formulas of the previous subsection has at most one fractional variable, the one associated to the split item. **Branching can be performed on the variable associated to the split item**: this branching-variable selection strategy is called split-item strategy.
- The optimal solution of an LP_{KP} problem is not unique when some items have equal efficiency: the other optimal solutions can have more fractional variables, for instance $x = (\frac{1}{2}, \frac{1}{2})$ for $p = (2, 2)$, $w = (2, 2)$, $c = 2$. Here, as in the general algorithm, we always use the solution given by the formulas.

41.1.3 What changes with respect to the general algorithm

- The algorithm is the LP-based branch-and-bound algorithm of the module *Integer linear programming* (Chapter 7 «Branch-and-bound»), with its problem list, its pruning criteria and its optimality gap.
- **Bounding.** The problem of a node is solved by the formulas above. All the profits are integer, so the bound of a node is $\lfloor z(\text{LP}_{\text{KP}}^k) \rfloor$: the node is pruned by bound if $\lfloor z(\text{LP}_{\text{KP}}^k) \rfloor \leq \text{LB}$, by infeasibility if $c(k) < 0$, by integrality if its solution is integer (then LB is updated when the value is larger). An integer node is pruned by integrality even when its value does not exceed LB.
- **Branching.** The branching variable is the one of the split item s : the children are $x_s \leq 0$ (down, item unselected) and $x_s \geq 1$ (up, item selected), and the node is not solved with a simplex tableau but described by the sets $\mathcal{J}_{\text{in}}^k$, $\mathcal{J}_{\text{out}}^k$ and $\mathcal{J}^k$.
- **Node selection.** The node of the problem list is extracted with the FIFO or the LIFO rule, and the down-branching node is solved before the up-branching node; the nodes are numbered in the order of resolution. The lower bound written in a node is the value at the moment of its resolution.

41.2 A worked example

Example 41.1: The ILP_{KP} model

We consider a knapsack of capacity $c = 9$ and a set $\mathcal{J} = \{1, 2, 3, 4, 5, 6\}$ of 6 items ($n = 6$), with the following profits and weights:

	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
p_j	6	5	8	9	6	7
w_j	2	3	6	7	5	9

We refer to this KP instance as [Example 41.1](#). The ILP_{KP} model for this KP instance is:

$$\begin{aligned}
 \max \quad & 6x_1 + 5x_2 + 8x_3 + 9x_4 + 6x_5 + 7x_6 \\
 \text{subject to} \quad & 2x_1 + 3x_2 + 6x_3 + 7x_4 + 5x_5 + 9x_6 \leq 9 \\
 & x_1, x_2, x_3, x_4, x_5, x_6 \in \{0, 1\}
 \end{aligned}$$

An optimal solution $\tilde{\mathbf{x}}(\text{ILP}_{\text{KP}})$ is:

$$\tilde{x}_1 = 1, \tilde{x}_2 = 0, \tilde{x}_3 = 0, \tilde{x}_4 = 1, \tilde{x}_5 = 0, \tilde{x}_6 = 0$$

The optimal value $z(\text{ILP}_{\text{KP}})$ is 15. The associated optimal solution of the KP is $\tilde{S} = \{1, 4\}$.

The items are sorted by non-increasing efficiency, since:

$$\frac{6}{2} \geq \frac{5}{3} \geq \frac{8}{6} \geq \frac{9}{7} \geq \frac{6}{5} \geq \frac{7}{9}$$

Example 41.2: Branch-and-bound algorithm for the ILP_{KP} model

A branch-and-bound algorithm for the ILP_{KP} model and the KP instance of [Example 41.1](#) using:

1. the split-item strategy as branching-variable selection strategy;
2. the FIFO rule as branching-node selection strategy;
3. solve first the down-branching node and then the up-branching node;
4. set the incumbent solution $\bar{x}(ILP_{KP})$ to the following heuristic solution:

$$\bar{x}_1 = 1, \quad \bar{x}_2 = 0, \quad \bar{x}_3 = 0, \quad \bar{x}_4 = 1, \quad \bar{x}_5 = 0, \quad \bar{x}_6 = 0$$

and set the lower bound LB accordingly.

- **Initialization**

For the LP_{KP}^1 problem, the split item is $s = 3$ and the residual capacity is $rc = 4$. An optimal solution $\tilde{x}(LP_{KP}^1)$ is:

$$\tilde{x}_1 = 1, \quad \tilde{x}_2 = 1, \quad \tilde{x}_3 = \frac{2}{3}, \quad \tilde{x}_4 = 0, \quad \tilde{x}_5 = 0, \quad \tilde{x}_6 = 0$$

the optimal value is:

$$z(LP_{KP}^1) = 6 \cdot 1 + 5 \cdot 1 + 8 \cdot \frac{2}{3} = \frac{49}{3}$$

The optimal solution $\tilde{x}(LP_{KP}^1)$ is **fractional** and the LP_{KP}^1 problem is **feasible**, so the LP_{KP}^1 problem is inserted in the problem list L and the lower bound LB is set to 15 ($LB(\bar{x}) = 15$).

Due to the current lower bound LB and upper bound UB, we have:

$$15 \leq z(ILP_{KP}) \leq \underbrace{\left\lfloor \frac{49}{3} \right\rfloor}_{=16}$$

- **Iteration 1**

The LP_{KP}^1 problem is extracted and deleted from the problem list L .

The branching variable is x_3 . The LP_{KP}^2 and LP_{KP}^3 problems are created by unselecting ($x_3 \leq 0$) and selecting ($x_3 \geq 1$) item 3.

1. Solution of the LP_{KP}^2 problem

We have

$$\mathcal{J}^2 = \{1, 2, 4, 5, 6\}, \quad \mathcal{J}_{out}^2 = \{3\}, \quad \mathcal{J}_{in}^2 = \emptyset$$

The capacity is $c(2) = 9$ and the objective-function constant is $p(2) = 0$. The split item is $s = 4$ and the residual capacity is $rc = 4$. An optimal solution $\tilde{\mathbf{x}}(LP_{KP}^2)$ is:

$$\tilde{x}_1 = 1, \quad \tilde{x}_2 = 1, \quad \tilde{x}_4 = \frac{4}{7}, \quad \tilde{x}_5 = 0, \quad \tilde{x}_6 = 0$$

and

$$z(LP_{KP}^2) = 6 \cdot 1 + 5 \cdot 1 + 9 \cdot \frac{4}{7} = \frac{113}{7}$$

The LP_{KP}^2 problem is **feasible** and we have:

$$\lfloor z(LP_{KP}^2) \rfloor > LB \quad \text{since} \quad \underbrace{\left\lfloor \frac{113}{7} \right\rfloor}_{=16} > 15$$

The optimal solution $\tilde{\mathbf{x}}(LP_{KP}^2)$ is **fractional**, accordingly, the LP_{KP}^2 problem is **inserted in the problem list L** .

 2. Solution of the LP_{KP}^3 problem

We have

$$\mathcal{J}^3 = \{1, 2, 4, 5, 6\}, \quad \mathcal{J}_{out}^3 = \emptyset, \quad \mathcal{J}_{in}^3 = \{3\}$$

The capacity is $c(3) = 3$ and the objective-function constant is $p(3) = 8$. The split item is $s = 2$ and the residual capacity is $rc = 1$. An optimal solution $\tilde{\mathbf{x}}(LP_{KP}^3)$ is:

$$\tilde{x}_1 = 1, \quad \tilde{x}_2 = \frac{1}{3}, \quad \tilde{x}_4 = 0, \quad \tilde{x}_5 = 0, \quad \tilde{x}_6 = 0 \quad \text{and} \quad z(LP_{KP}^3) = 8 + 6 \cdot 1 + 5 \cdot \frac{1}{3} = \frac{47}{3}$$

The LP_{KP}^3 problem is **feasible** and we have:

$$\lfloor z(LP_{KP}^3) \rfloor \leq LB \quad \text{since} \quad \underbrace{\left\lfloor \frac{47}{3} \right\rfloor}_{=15} \leq 15$$

accordingly, the branching node is pruned by upper bound.

At the end of the first iteration, Node 2 is inserted in the problem list L , while Node 3 is pruned by upper bound. The corresponding branch-and-bound tree is:

Node 1

$$\begin{array}{l}
 \max \quad 6x_1 + 5x_2 + 8x_3 + 9x_4 + 6x_5 + 7x_6 \\
 2x_1 + 3x_2 + 6x_3 + 7x_4 + 5x_5 + 9x_6 \leq 9 \\
 x_1, x_2, x_3, x_4, x_5, x_6 \leq 1 \\
 x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \\
 \tilde{x}_1 = 1, \tilde{x}_2 = 1, \tilde{x}_3 = \frac{2}{3}, \tilde{x}_4 = 0, \tilde{x}_5 = 0, \tilde{x}_6 = 0 \\
 \lfloor z(\text{LP}_{\text{KP}}^1) \rfloor = \underbrace{\left\lfloor \frac{49}{3} \right\rfloor}_{=16} \\
 \text{LB} = 15
 \end{array}$$

Node 2

$$x_3 \leq 0$$

$$\begin{array}{l}
 \max \quad 6x_1 + 5x_2 + 9x_4 + 6x_5 + 7x_6 \\
 2x_1 + 3x_2 + 7x_4 + 5x_5 + 9x_6 \leq 9 \\
 x_1, x_2, x_4, x_5, x_6 \leq 1 \\
 x_1, x_2, x_4, x_5, x_6 \geq 0 \\
 \tilde{x}_1 = 1, \tilde{x}_2 = 1, \tilde{x}_4 = \frac{4}{7}, \tilde{x}_5 = 0, \tilde{x}_6 = 0 \\
 \lfloor z(\text{LP}_{\text{KP}}^2) \rfloor = \underbrace{\left\lfloor \frac{113}{7} \right\rfloor}_{=16} \\
 \text{LB} = 15
 \end{array}$$

Node 3

$$x_3 \geq 1$$

$$\begin{array}{l}
 8 + \max \quad 6x_1 + 5x_2 + 9x_4 + 6x_5 + 7x_6 \\
 2x_1 + 3x_2 + 7x_4 + 5x_5 + 9x_6 \leq 3 \\
 x_1, x_2, x_4, x_5, x_6 \leq 1 \\
 x_1, x_2, x_4, x_5, x_6 \geq 0 \\
 \tilde{x}_1 = 1, \tilde{x}_2 = \frac{1}{3}, \tilde{x}_4 = 0, \tilde{x}_5 = 0, \tilde{x}_6 = 0 \\
 \lfloor z(\text{LP}_{\text{KP}}^3) \rfloor = \underbrace{\left\lfloor \frac{47}{3} \right\rfloor}_{=15} \\
 \text{LB} = 15 \\
 \text{pruned by upper bound}
 \end{array}$$

Due to the current lower bound LB and upper bound UB, we have:

$$15 \leq z(\text{ILP}_{\text{KP}}) \leq \underbrace{\left\lfloor \frac{113}{7} \right\rfloor}_{=16}$$

• Iteration 2

The LP_{KP}^2 problem is extracted and deleted from the problem list L .

The branching variable is x_4 . The LP_{KP}^4 and LP_{KP}^5 problems are created by unselecting ($x_4 \leq 0$) and selecting ($x_4 \geq 1$) item 4.

 1. Solution of the LP_{KP}^4 problem

We have

$$\mathcal{J}^4 = \{1, 2, 5, 6\}, \quad \mathcal{J}_{\text{out}}^4 = \{3, 4\}, \quad \mathcal{J}_{\text{in}}^4 = \emptyset$$

The capacity is $c(4) = 9$ and the objective-function constant is $p(4) = 0$. The split item is $s = 5$ and the residual capacity is $rc = 4$. An optimal solution $\tilde{x}(\text{LP}_{\text{KP}}^4)$ is:

$$\tilde{x}_1 = 1, \quad \tilde{x}_2 = 1, \quad \tilde{x}_5 = \frac{4}{5}, \quad \tilde{x}_6 = 0 \quad \text{and} \quad z(\text{LP}_{\text{KP}}^4) = 6 \cdot 1 + 5 \cdot 1 + 6 \cdot \frac{4}{5} = \frac{79}{5}$$

The LP_{KP}^4 problem is **feasible** and we have:

$$\lfloor z(\text{LP}_{\text{KP}}^4) \rfloor \leq \text{LB} \quad \text{since} \quad \underbrace{\left\lfloor \frac{79}{5} \right\rfloor}_{=15} \leq 15$$

accordingly, the branching node is pruned by upper bound.

2. Solution of the LP_{KP}^5 problem

We have

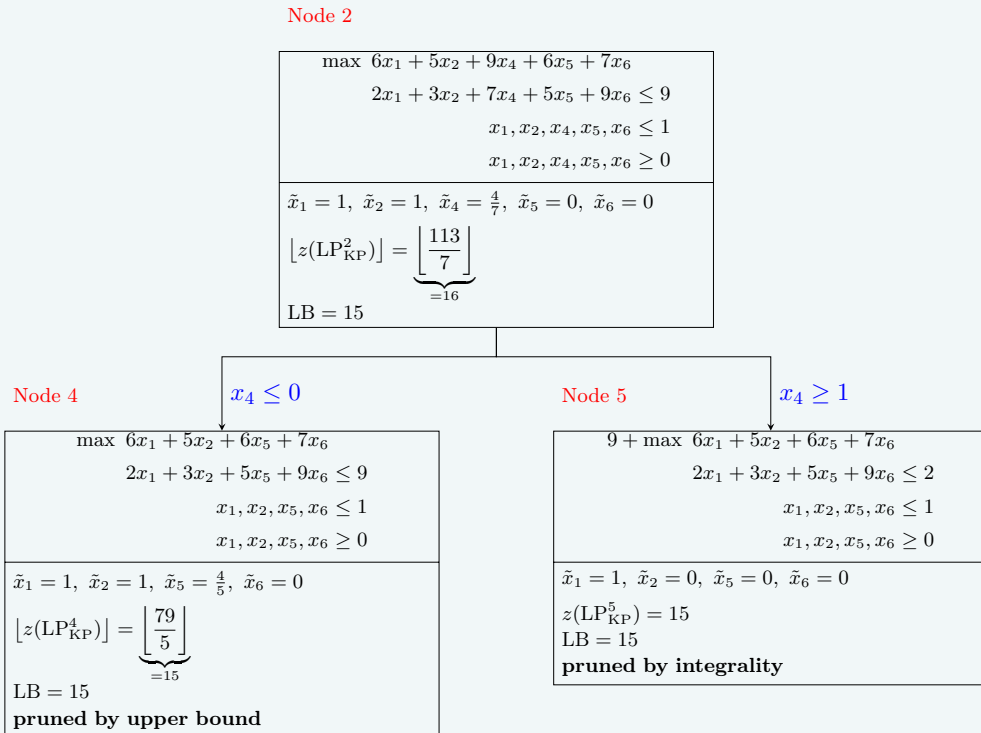
$$\mathcal{J}^5 = \{1, 2, 5, 6\}, \quad \mathcal{J}_{\text{out}}^5 = \{3\}, \quad \mathcal{J}_{\text{in}}^5 = \{4\}$$

The capacity is $c(5) = 2$ and the objective-function constant is $p(5) = 9$. The split item is $s = 2$ and the residual capacity is $rc = 0$. An optimal solution $\tilde{x}(\text{LP}_{\text{KP}}^5)$ is:

$$\tilde{x}_1 = 1, \quad \tilde{x}_2 = 0, \quad \tilde{x}_5 = 0, \quad \tilde{x}_6 = 0 \quad \text{and} \quad z(\text{LP}_{\text{KP}}^5) = 9 + 6 \cdot 1 = 15$$

The LP_{KP}^5 problem is **feasible** and its optimal solution $\tilde{x}(\text{LP}_{\text{KP}}^5)$ is **integer**. Its value does not improve the incumbent solution, since $z(\text{LP}_{\text{KP}}^5) = 15 \leq 15 = \text{LB}$, so the lower bound LB stays 15. Accordingly, the branching node is pruned by integrality: when a node could be pruned both by integrality and by bound, it is pruned by integrality.

At the end of the second iteration, Node 4 is pruned by upper bound and Node 5 is pruned by integrality. The corresponding branch-and-bound tree is:

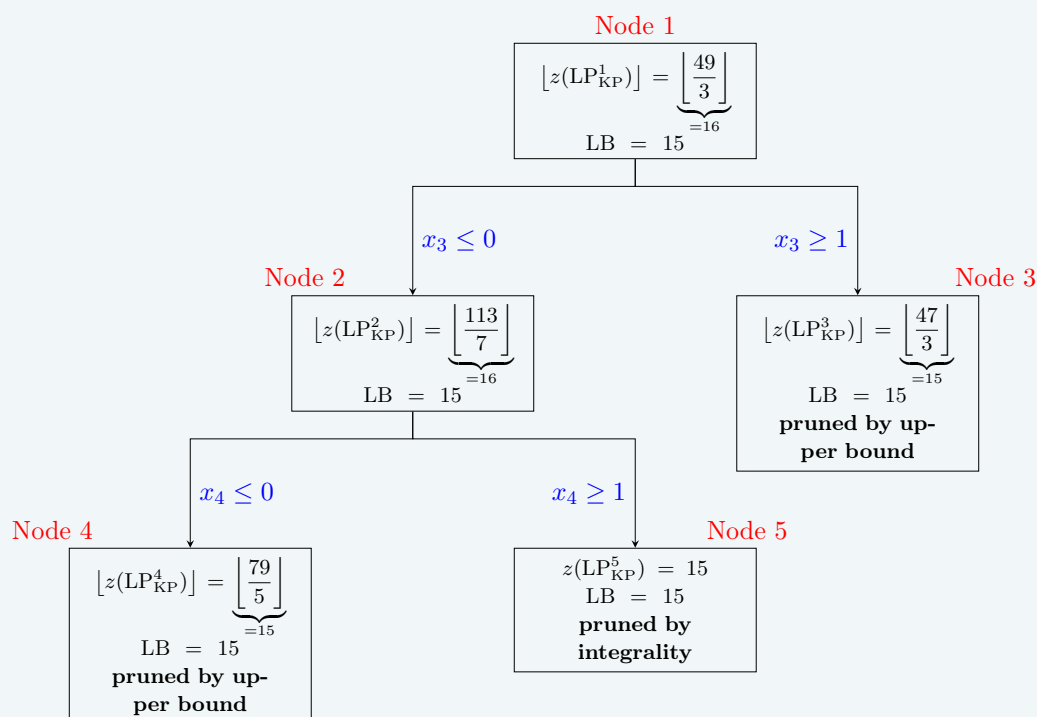


- **Termination**

Since the problem list L is empty, the ILP_{KP} problem is solved to optimality.

The optimal value is $z(\text{ILP}_{\text{KP}}) = 15$ (the value of the lower bound LB) and an optimal solution $\tilde{\mathbf{x}}(\text{ILP}_{\text{KP}})$ is $\tilde{x}_1 = 1, \tilde{x}_2 = 0, \tilde{x}_3 = 0, \tilde{x}_4 = 1, \tilde{x}_5 = 0, \tilde{x}_6 = 0$ (the incumbent solution $\bar{\mathbf{x}}(\text{ILP}_{\text{KP}})$).

This is the branch-and-bound tree of the previous execution of the branch-and-bound algorithm, showing for each branching node the optimal value of the associated LP_{KP} problem, the lower bound LB, and the pruning criteria.



41.3 Typical mistakes

Warning

- **Forgetting the upper bounds $x_j \leq 1$.** Without them the LP problem of a node would put all the capacity on the item of best efficiency: the node boxes must contain $x_j \leq 1$.
- **Computing the split item on the original instance.** In a node, the split item and the residual capacity are computed on the items of $\mathcal{J}^k$ with the capacity $c(k)$, after fixing the items of $\mathcal{J}_{\text{in}}^k$ and $\mathcal{J}_{\text{out}}^k$.
- **Forgetting the constant $p(k)$.** The value of the node is $p(k)$ plus the value of the items of $\mathcal{J}^k$: the profit of the selected items counts.
- **Using the old lower bound.** The lower bound is updated as soon as a node has an integer solution of larger value; the nodes of the list that become dominated are discarded.
- **Numbering the skipped children.** A child that is not solved does not receive a node index, exactly as in the general algorithm.

41.4 In summary

Summary

- The branch-and-bound algorithm for the KP is the LP-based algorithm with two simplifications: the problem of every node is solved by the formulas of Dantzig, and the branching variable is the one of the split item.
- A node is described by the sets $\mathcal{J}_{\text{in}}^k$, $\mathcal{J}_{\text{out}}^k$ and $\mathcal{J}^k$, by the capacity $c(k)$ and by the constant $p(k)$.
- The bound of a node is $\lfloor z(\text{LP}_{\text{KP}}^k) \rfloor$; when the problem list is empty the lower and the upper bound coincide and the incumbent solution is optimal.

Dashboard of the KP instance of Example 41.1 ($c = 9$, $p = (6, 5, 8, 9, 6, 7)$, $w = (2, 3, 6, 7, 5, 9)$). The bounds on $z(\text{ILP}_{\text{KP}})$ found so far:

Source	Lower bound	Upper bound	From
Dantzig bound	–	16	$\lfloor \frac{49}{3} \rfloor$
This chapter, initialization	15	16	heuristic solution $\{1, 4\}$
This chapter, iteration 1	15	16	node 2 in the list, $\lfloor \frac{113}{7} \rfloor$
This chapter, termination	15	15	empty list, 5 nodes

Coloring problems

Vertex coloring: the DSATUR branch-and-bound

Prerequisites

- From the module *Integer linear programming*, [Chapter 7 «Branch-and-bound»](#): branching and bounding operations, the problem list, the pruning criteria, the bounds LB and UB.
- From this module, [Chapter 4 «Coloring problems»](#): the vertex coloring problem, cliques, stable sets and the clique bound $\omega(G) \leq \chi(G)$.
- From this module, [Chapter 30 «Constructive heuristics»](#): the saturation degree and the *DSATUR* heuristic.

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.

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 $\chi(G)$.

42.1 The algorithm for the VCP

Definition 42.1: 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 $Q_c^k := \{v \in V^k : col^k(v) = c\}$, $c = 1, \dots, q(k)$. For an uncolored vertex $v \in U^k$:

$$sat_k(v) := |\{col^k(u) : u \in N(v) \cap V^k\}| \quad (\text{saturation degree}),$$

$$du_k(v) := |N(v) \cap U^k| \quad (\text{degree among the uncolored vertices}),$$

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

- The saturation degree counts the *different* colors among the colored neighbours of v , and $|F_k(v)| = q(k) - sat_k(v)$.
- A node with $U^k = \emptyset$ is a **complete coloring**: a feasible solution of the VCP with $q(k)$ colors.

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.

42.1.1 Branching

- **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).
- **Children.** 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 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.

42.1.2 Bounds and pruning

- 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 Chapter 30 «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.** 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.** 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.

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.

42.1.3 Node selection and pseudocode

- 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.

Algorithm

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

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

Step 0: Initialization

- Run *DSATUR*: set the incumbent solution to its coloring and UB to its number of colors.
- Find the greedy clique $K = \{k_1, \dots, k_r\}$ and set $LB \leftarrow r$. If $LB = UB$, the incumbent solution is optimal. **STOP**.
- 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 $c \leftarrow 1$.

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.

- 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.
- This is the pseudocode of the **node sub-procedure**.

Algorithm

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

Step 1: Bounding operations

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

42.1.4 Correctness
Proposition 42.2: 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.

Proof. 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 .

1. **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.

- 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 $\mathcal{Q}$.

2. **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. $\square$

- **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.

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.

42.2 A worked example

Example 42.1: DSATUR branch-and-bound on the Petersen graph

We consider the Petersen graph, the reference instance of [Chapter 4 «Coloring problems»](#), with $n = 10$ vertices, $m = 15$ edges and all degrees equal to 3:

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

The DSATUR branch-and-bound algorithm uses the rules of Section 42.1: branching on the most saturated vertex, depth-first search, children in increasing order of color.

- **Initialization**

1. The *DSATUR* heuristic (Example «*DSATUR* on the Petersen graph» of [Chapter 30](#)) colors the vertices in the order 1, 2, 3, 4, 5, 6, 8, 7, 9, 10 and gives the incumbent solution

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

2. All the degrees are equal, so the greedy clique scans the vertices 1, 2, ..., 10. Vertex 1 enters, vertex 2 is adjacent to 1 and enters, and no other vertex is adjacent to both. Hence $K = \{1, 2\}$ and $\text{LB} = 2$. The Petersen graph has no triangle, so $\omega(G) = 2$.
3. Node 1 colors $\text{col}^1(1) = 1$ and $\text{col}^1(2) = 2$, with $q(1) = 2$. Since $\text{LB} = 2 < 3 = \text{UB}$, node 1 is inserted into the list L :

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

- **Iteration 1**

Node 1 is extracted from L . The vertices 3, 5, 6, 7 have one colored neighbour each, so $\text{sat}_1 = 1$; the others have $\text{sat}_1 = 0$. The four tied vertices have $\text{du}_1 = 2$, so the smallest index wins: the branching vertex is 3. Its neighbour 2 has color 2, so $F_1(3) = \{1\}$.

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

- **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 $\text{sat}_2 = 1$ and $\text{du}_2 = 2$, and 9, 10 have $\text{sat}_2 = 0$: the branching vertex is 4. Its neighbour 3 has color 1, so $F_2(4) = \{2\}$.

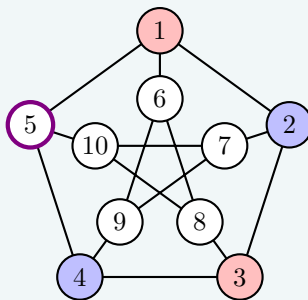
1. Node 4: $\text{col}(4) = 2$, with $q(4) = 2 < \text{UB}$. The node survives.
2. Node 5: $\text{col}(4) = 3$, with $q(5) = 3 \geq \text{UB}$. The node is pruned by bound.

• Iteration 3

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): $\text{sat}_4(5) = 2$, the only vertex with saturation degree 2, so it is the branching vertex. Its feasible colors are $F_4(5) = \emptyset$.

1. Node 6: $\text{col}(5) = 3$, the only child, with $q(6) = 3 \geq \text{UB}$. The node is pruned by bound.

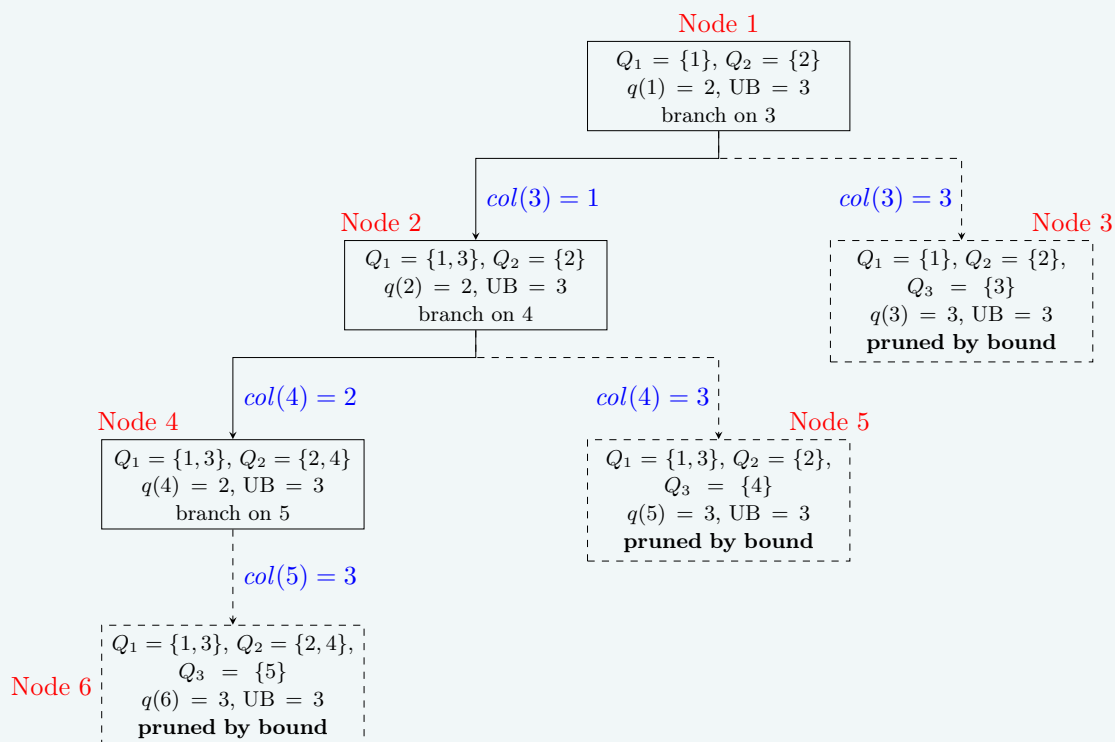
The partial coloring of node 4 is drawn below: the cycle 1, 2, 3, 4, 5 has five vertices, and two colors are not enough to close it.



• Termination

The list L is empty: the incumbent solution $\bar{Q}$ is optimal and $\chi(G) = \text{UB} = 3$. The tree has 6 nodes.

This is the branch-and-bound tree, showing for each node its color classes, the number of colors q , the upper bound UB at the moment of the evaluation, and the branching vertex or the pruning criterion. 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

- 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 [Chapter 4](#), as Proposition 42.2 predicts.

42.3 Remarks

- **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 Proposition 42.2 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.

42.4 Typical mistakes

Warning

- **Counting colored neighbours instead of colors.** The saturation degree counts the *different* colors around v : two neighbours with the same color count once.
- **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.
- **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.
- **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 $q(k) \geq \text{UB}$.
- **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.
- **Continuing after** $\text{UB} = |K|$. When a complete coloring reaches the clique bound, it is optimal: the nodes still in the list are not explored.

42.5 In summary

Summary

- 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.

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 (Chapter 4)	2	–	$\omega(G) = 2$
<i>DSATUR</i> (Chapter 30)	–	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