Data-driven Decision Making Integer optimization, Location problems

João Pedro Pedroso

2024/2025

João Pedro Pedroso

Data-driven Decision Making

2024/2025

1/80

• Determining the sites for factories and warehouses

- modeling:
 - demand satisfaction
 - capacity constraints
- Capacitated facility location
 - total demand that each facility may satisfy is limited
- Uncapacitated facility location
 - $\bullet\,$ no limits on total demand $\rightarrow\,$ big M
- Weak and strong formulations

Different formulations



2024/2025

Ideal formulation



• More Facility Location Problems

- k-Median
- k-Center
- k-Cover
- Graph problems

47 ▶

The k-Median problem



Median problem

Select a given number of facilities from possible points in a graph, in such a way that the sum of the distances from each customer to the closest facility is minimized

- Often, the number *k* of facilities to be selected is predetermined in advance
- *k* median problem:
 - variant of uncapacitated facility location problem
 - seeks to establish k facilities without considering fixed costs
 - each demand point served by exactly one facility
 - objective: serve all demand points at minimum total cost

The k-Median problem

- Notation:
 - distance from customer i to facility $j
 ightarrow c_{ij}$
 - set of customers $ightarrow \{1,\ldots,n\}$
 - set of potential places for facilities $ightarrow \{1,\ldots,m\}$
 - commonly, facilities and customers share the same set of points
- Variables:

$$x_{ij} = \begin{cases} 1 & \text{when the demand of customer } i \text{ is met by facility } j \\ 0 & \text{otherwise} \end{cases}$$

$$y_j = \left\{ egin{array}{cc} 1 & ext{when facility } j ext{ is open} \ 0 & ext{otherwise} \end{array}
ight.$$

The k-Median problem

minimize

subject to:

$$\sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}$$

$$\sum_{j=1}^{m} x_{ij} = 1 \text{ for } i = 1, \cdots, n$$

$$\sum_{j=1}^{m} y_j = k$$

$$x_{ij} \le y_j \text{ for } i = 1, \cdots, n; j = 1, \cdots, m$$

$$x_{ij} \in \{0, 1\} \text{ for } i = 1, \cdots, n; j = 1, \cdots, m$$

$$y_j \in \{0, 1\} \text{ for } j = 1, \cdots, m$$

< 47 ▶

æ

• each customer *i* is assigned to exactly one facility *j*

$$\sum_{j=1}^m x_{ij} = 1 \text{ for } i = 1, \cdots, n$$

• exactly k facilities are established

$$\sum_{j=1}^m y_j = k$$

• force facility *j* to be open if it serves demand point *i*

$$x_{ij} \leq y_j$$
 for $i = 1, \cdots, n; j = 1, \cdots, m$

• weaker formulation is obtained if we replace these *nm* constraints by *n* constraints

$$\sum_{i=1}^n x_{ij} \leq ny_j, ext{ for } j = 1, \cdots, m$$

 \rightarrow lead to worse values in the linear relaxation.

João Pedro Pedroso

Data-driven Decision Making

```
param n; # number or customers
param m; # number or facilities
param c {1..n, 1..m};
param k;
var x {1..n, 1..m} binary;
var y {1..m} binary;
minimize cost: sum {i in 1..n, j in 1..m} c[i,j] * x[i,j];
subject to
Serve {i in 1..n}: sum {j in 1..m} x[i,j] = 1;
Kfacil: sum {j in 1..m} y[j] = k;
Activate {i in 1..n, j in 1..m}: x[i,j] <= y[j];</pre>
```

= nar

11 / 80

< ロ > < 同 > < 回 > < 回 > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ >

- Solution obtained for a graph with 200 vertices placed randomly in the two-dimensional unit box
- Costs given by Euclidean distance
- Each of the vertices is a potential location for a facility



æ

- AMPL model: see above
- AMPL data: replaced by a Python program
- $\rightarrow \texttt{see file kmedian_plt.py}$

э

The k-Center Problem

João Pedro Pedroso

16 / 80

Center problem

Select a given number of facilities from possible points in a graph, in such a way that the maximum value of a distance from a customer to the closest facility is minimized.

- Variant of the k-median problem
- Assign facilities to a subset of vertices
 - aim: each customer "close" to some facility
- Number k of facilities is predetermined

The k-Center problem

- Notation (same as k-Median):
 - distance from customer *i* to facility $\rightarrow j c_{ij}$
 - set of customers $ightarrow \{1,\ldots,n\}$
 - set of potential places for facilities $ightarrow \{1,\ldots,m\}$
 - commonly, facilities and customers share the same set of points
- Variables (same as k-Median):

 $\begin{aligned} x_{ij} &= \left\{ \begin{array}{ll} 1 & \text{when the demand of customer } i \text{ is met by facility } j \\ 0 & \text{otherwise} \end{array} \right. \\ y_j &= \left\{ \begin{array}{ll} 1 & \text{when facility } j \text{ is open} \\ 0 & \text{otherwise} \end{array} \right. \end{aligned}$

Distance/cost for most distant customer from an activated facility
 additional continuous variable z

The k-Center problem

minimize 7 subject to: $\sum_{i=1}^{m} x_{ij} = 1$ for $i = 1, \cdots, n$ $\sum_{j=1}^m y_j = k$ $x_{ii} \leq y_i$ for $i = 1, \dots, n; j = 1, \dots, m$ $\sum_{j=1} c_{ij} x_{ij} \le z$ for $i = 1, \cdots, n$ $x_{ii} \in \{0, 1\}$ for $i = 1, \dots, n; j = 1, \dots, m$ $v_i \in \{0, 1\}$ for $i = 1, \cdots, m$

2024/2025

• New constraint:

$$\sum_{j=1}^m c_{ij} x_{ij} \leq z \quad \text{ for } i = 1, \cdots, n$$

- Determine z to take on at least c_{ij}
 - for all facilities j and customers i assigned to j
 - weaker (maybe more natural) version:

$$c_{ij}x_{ij} \leq z$$
, for $i = 1, \cdots, n; j = 1, \cdots, m$

- intuition: in the strong formulation we are adding more terms in the left-hand side \rightarrow feasible region is tighter
- New objective: z
 - $\bullet\,$ minimizing a maximum value $\rightarrow\,$ min-max objective
 - type of problems for which mathematical optimization solvers are typically weak
 - instead of the previous objective

minimize

$$\sum_{i=1}^n \sum_{j=1}^m c_{ij} x_{ij}$$

20 / 80

```
param n; # number or customers
param m; # number or facilities
param c {1...n, 1...m};
param k;
var x {1...n, 1...m} binary;
var y {1..m} binary;
var z \ge 0:
minimize maxcost: z;
subject to
Serve {i in 1..n}: sum {j in 1..m} x[i,j] = 1;
Kfacil: sum {j in 1..m} y[j] = k;
Activate {i in 1..n, j in 1..m}: x[i,j] <= y[j];
MinZ {i in 1..n}: sum {j in 1..m} c[i,j] * x[i,j] <= z;
```

- Observe difference in performance
- Observe difference in the solution

- "Minimization of the maximum value" can be reduced to a standard linear optimization
 - add new variable
 - make that variable at least as large as each of the values
- Assume that we want to minimize the maximum of two linear expressions:
 - $3x_1 + 4x_2$
 - $2x_1 + 7x_2$.
 - minimize new variable z subject to:

 $3x_1 + 4x_2 \le z$ $2x_1 + 7x_2 \le z$

Minimization of the absolute value |x| of a real variable x:

- Nonlinear expression
- To linearaize it: add non-negative variables y and z
 - $x = y z \rightarrow$ value of x in terms of y and z
 - now, |x| can be written as y + z
- So:
 - occurrences of x in the formulation \rightarrow replaced by y z
 - |x| in the objective function \rightarrow replaced by y + z.
- Another possibility:
 - adding variable z
 - impose $z \ge x$ and $z \ge -x$
 - z replaces |x| in the objective function

An objective function that minimizes a maximum value should be avoided, if possible.

- ullet In integer optimization o solved by the branch-and-bound method
- If the objective function minimizes the maximum value of a set of variables:
 - tendency to have large values for the difference between the lower bound and the upper bound (the so-called duality gap).
 - time for solving the problem becomes large
 - if branch-and-bound is interrupted, the incumbent solution is rather poor.

The k-Cover Problem

João Pedro Pedroso

→

• • • • • • • • •

- Variant to k-center problem
- Avoids the min-max objective
- Process makes use of binary search

- Graph $G_{\theta} = (V, E_{\theta})$
 - set of edges whose distances from a customer to a facility which do not exceed a threshold value θ
 - edges: $E_{\theta} = \{\{i, j\} \in E : c_{ij} \leq \theta\}$
- Subset S ⊆ V is called a cover if every vertex i ∈ V is adjacent to at least one of the vertices in S
- Idea: optimum value of the k-center problem ≤ θ if there exists a cover with cardinality |S| = k on graph G_θ.

28 / 80

• Variables:

- $y_j = 1$ if a facility is opened at j, 0 otherwise
 - vertex *j* is in the subset *S* or not
- z_i = 1 if vertex i is adjacent to no vertex in S, 0 otherwise
 vertex i is covered or not
- $[a_{ij}] \rightarrow$ incidence matrix of $G_{ heta}$
 - $a_{ij} = 1$ if vertices *i* and *j* are adjacent, 0 otherwise
- We need to determine whether or not graph $G_{ heta}$ has a cover |S| = k
- We can do that by solving integer-optimization model \rightarrow k-cover problem on ${\cal G}_{\theta}$

The k-Cover Problem: model

$$\begin{array}{ll} \text{minimize } \sum_{i=1}^{n} z_i \\ \text{subject to } \sum_{j=1}^{m} a_{ij} y_j + z_i \geq 1 & \text{for } i = 1, \cdots, n \\ & \sum_{j=1}^{m} y_j = k \\ & z_i \in \{0, 1\} & \text{for } i = 1, \cdots, n \\ & y_j \in \{0, 1\} & \text{for } j = 1, \cdots, m. \end{array}$$

- adjacency matrix is built upon a given value of distance θ
 - used to compute set of facilities that may serve each of the customers within that distance

João Pedro Pedroso

2024/2025

30 / 80

• Given θ , either:

- optimal objective value of the previous optimization problem is zero
 - k facilities were enough for covering all the customers withing distance θ
 - $\bullet \ \rightarrow {\rm reduce} \ \theta$
- greater that zero
 - there is at least one $z_i > 0$
 - thus, a customer could not be served from any of the k open facilities
 - \rightarrow increase θ
- binary search: repeate until bounds for θ are close enough

31 / 80

```
param n; # number or customers
param m; # number or facilities
param c {1..n, 1..m};
param k;
param theta;
var y {1..m} binary;
var z {1..n} binary;
minimize cost: sum {i in 1..n} z[i];
subject to
Serve {i in 1..n}:
    sum {j in 1..m : c[i,j] <= theta} y[j] + z[i] >= 1;
Kfacil: sum {j in 1..m} y[j] = k;
```

イロト イ理ト イヨト イヨト

= nar

Programming: loading the model

```
random.seed(1)
n = 200
m = n
I,J,c,x_pos,y_pos = make_data(n,m)
k = 3
ampl = AMPL()
ampl.option['solver'] = 'gurobi'
ampl.read("kcover.mod")
ampl.param['n'] = n
ampl.param['m'] = n
ampl.param['k'] = k
ampl.param['c'] = c
print("solving")
start = time.time()
facilities,edges = [],[]
delta = 1.e-4 # tolerance
LB = 0
UB = max(c[i,j] \text{ for } (i,j) \text{ in } c)
```

< □ > < 同 > < 回 > < 回 > < 回 >

э

```
delta = 1.e-4 # tolerance
I.B = 0
UB = max(c[i,j] \text{ for } (i,j) \text{ in } c)
while UB-LB > delta:
    theta = (UB+LB) / 2.
    ampl.param['theta'] = theta
    ampl.solve()
    cost = ampl.pbj['cost']
    if cost.value() < 0.5:
        UB = theta
        y_{-} = ampl.var['y']
        facilities = [j for j in J if y_[j].value() > .5]
        edges = [(i,j) for i in I for j in facilities if c[i,j] < theta]
    else: # infeasibility > 0:
        LB = theta
```

3

A (1) < A (1) < A (1) </p>

- $\bullet\,$ k-cover within binary search $\to\,$ time comparable k-median check
- in practice, k-center solution is usually preferable to k-median
 - longest time required for serving a customer
 - may be large on the k-median solution.

Using k-Cover and binary search: $\theta = 0$ (redundant)



João Pedro Pedroso

Data-driven Decision Making

2024/2025
Using k-Cover and binary search: $\theta = \max c_{ij}$ (redundant)





2024/2025



João Pedro Pedroso

Data-driven Decision Making

2024/2025

39 / 80



LB:0.311, theta:0.388, UB:0.466



LB:0.388, theta:0.427, UB:0.466



LB:0.427, theta:0.446, UB:0.466



LB:0.427, theta:0.437, UB:0.446



LB:0.437, theta:0.442, UB:0.446



LB:0.442, theta:0.444, UB:0.446



LB:0.444, theta:0.445, UB:0.446



LB:0.444, theta:0.445, UB:0.445



Using k-Cover and binary search $\theta = 0.445+$ (final)

Final k-Cover Solution, theta=0.445

Using k-Cover and k-Center



k-Median Solution



< ロ > < 同 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ >



2024/2025

João Pedro Pedroso

・ロト ・ 四ト ・ ヨト ・ ヨト

This class:

- Based on the definition of a graph
 - abstract object composed of vertices and edges
 - $\bullet \ \ \mathsf{edge} \to \mathsf{link} \ \mathsf{between} \ \mathsf{two} \ \mathsf{vertices}$
 - useful tool to unambiguously represent many real problems
 - many practical optimization problems can be defined naturally with graphs
- Three optimization problems with a combinatorial structure:
 - graph coloring problem

Example: friendship relationship as a graph

- You have six friends
 - represent each of them by a circle
 - $\bullet\,$ in graph theory $\rightarrow\,$ called vertices, nodes or points
- Some of these fellows have a good relationship between them, others not
- To organize these relashionships:
 - connect with a line each pair of persons which are in good terms
 - $\bullet\,$ in graph theory \rightarrow called an edge, arc or line
- $\bullet\,$ This representation is a graph $\to\,$ friendship scenario becomes very easy to grasp



- Undirected graphs: lines connecting two vertices have no implied direction
 - $\bullet~$ lines with no direction $\rightarrow~edges$
- Directed graphs: direction of lines connecting two vertices is important
 - $\bullet~{\rm directed~lines} \to {\rm arcs}$
- Set of vertices: V
- Set of edges: E
- Graph: G = (V, E)
- Vertices which are endpoints of an edge: adjacent to each other
- Edge is incident to vertices at both ends
- Number of edges connected to a vertex: degree of the vertex.

João Pedro Pedroso

-

- 4 ⊒ →

You are concerned about how to assign a class to each of your friends. Those which are on unfriendly terms with each other are linked with an edge in the figure below. If put on the same class, persons on unfriendly terms will start a fight. To divide your friends into as few classes as possible, how should you proceed?



- Graph:
 - $\bullet \ \ \mathsf{nodes} \to \mathsf{persons}$
 - $\bullet\,$ edge between two persons $\rightarrow\,$ not good friends
 - three classes are enough to separate persons on unfriendly terms



- Graph:
 - $\bullet \ \ \mathsf{nodes} \to \mathsf{persons}$
 - $\bullet\,$ edge between two persons $\rightarrow\,$ not good friends
 - three classes are enough to separate persons on unfriendly terms



Map Coloring

- maps of countries
- adjacent coutries cannot have the same color
- four colors are sufficient ("Four Color Theorem")

Scheduling and timetabling

- different courses
- students enrolled in courses
- how to schedule exams so that no common students are scheduled at same time?
 - $\bullet \ \ \text{vertex} \to \text{course}$
 - $\bullet~\mbox{edge} \rightarrow \mbox{some student}$ on both courses

Mobile Radio Frequency Assignment

- frequencies are assigned to towers
- if towers are "close enough" \rightarrow interference \rightarrow frequencies must be different
 - $\bullet \ \ \text{tower} \to \text{vertex}$
 - $\bullet\,$ two towers are "close" $\rightarrow\,$ edge between them

Graph coloring: illustration



João Pedro Pedroso

Data-driven Decision Making

2024/2025

< 4[™] > <

60 / 80

표 문 표

Graph coloring: illustration



2024/2025

< 47 ▶

≣≯ ≣

- Given an undirected graph G = (V, E), a K partition is a division of the vertices V into K subsets V_1, V_2, \ldots, V_K such that
 - $V_i \cap V_j = \emptyset, \forall i \neq j \rightarrow \textit{no overlap}$
 - $\bigcup_{i=1}^{K} V_i = V \rightarrow$ union of subsets = full set of vertices
- Each V_i (i = 1, 2, ..., K) is called a color class.
- In a K partition, if all the vertices in a color class V_i forms a stable set, it is called K coloring → no edge among two vertices in that class

For a given undirected graph, the graph coloring problem consists of finding the minimum K for which there is a K coloring

- this is called the graph's chromatic number
- applications: timetabling, frequency allocation, ...

A (10) A (10)

- Requirement: upper bound K_{max} of the number of colors
- Optimal number of colors K: integer $1 \le K \le K_{\max}$
- Variables (binary):
 - $x_{ik} = 1$ if a vertex *i* is assigned color *k*, 0 otherwise
 - $y_k = 1$ if color k has been used, 0 otherwise
 - if $y_k = 1
 ightarrow$ set V_k contains at least one vertex
 - if $y_k = 0 \rightarrow V_k$ is empty (color k was not required)

Graph coloring problem: mathematical formulation

$$\begin{array}{ll} \text{minimize} & \sum_{k=1}^{K_{\text{max}}} y_k \\ \text{subject to} & \sum_{k=1}^{K_{\text{max}}} x_{ik} = 1 & \forall i \in V \\ & x_{ik} + x_{jk} \leq y_k & \forall \{i, j\} \in E; k = 1, \dots, K_{\text{max}} \\ & x_{ik} \in \{0, 1\} & \forall i \in V; k = 1, \dots, K_{\text{max}} \\ & y_k \in \{0, 1\} & k = 1, \dots, K_{\text{max}} \end{array}$$

- First constraint: exactly one color is assigned to each vertex
- Second constraint: connects variables x and y
 - allows coloring with color k only if $y_k = 1$
 - forbids the endpoints of any edge $\{i, j\}$ from having the same color simultaneously

João Pedro Pedroso

Remarks

- Most mathematical optimization solvers use branch-and-bound
- \bullet All color classes in the formulation above are treated indifferently \rightarrow solution space has a great deal of symmetry
- Symmetry causes troubles to branch-and-bound
 - increases enormously the size of the tree
 - e.g., solutions $V_1 = 1, 2, 3, V_2 = 4, 5$ and $V_1 = 4, 5, V_2 = 1, 2, 3$ are equivalent, but are represented by different vectors x and y
 - in this case, branching on any of the variables *x*, *y* leads to no improvements in the lower bound
- Avoiding symmetry in the graph coloring problem
 - $\bullet\,\rightarrow$ use preferentially color classes with low subscripts
 - may considerably improve solving time

$$y_k \ge y_{k+1}$$
 $k = 1, \dots, K_{\mathsf{max}} - 1$

When there is symmetry in a formulation, add constraints for removing it.

- When formulations for integer optimization problems have a large amount of symmetry, the branch-and-bound method is weak
- Adding constraints for explicitly breaking symmetry may dramatically improved solution time
- But there are no uniform guidelines...
 - Deciding what constraints should be added is not abvious
 - adding simple constraints such as those added in the graph coloring problem often works well
 - however, in some cases adding elaborate constraints breaks the structure of the problem, making the solver slower
- Careful experimentation is necessary for deciding if such constraints are useful

```
set V:
                               # vertices
set E within {V,V};
                              # edges
param Kmax default card(V); # upper bound to the number of colors
set C := {1..Kmax};
                             # set of colors
var x {V, C} binary;
var y {C} binary;
minimize K: sum {k in C} y[k];
subject to
Color {i in V}: sum {k in C} x[i,k] = 1;
Edge {(i,j) in E, k in C}: x[i,k] + x[j,k] \le y[k];
NoSym {k in 1...Kmax-1}: y[k] \ge y[k+1];
```

3

イロト イヨト イヨト -

How to instantiate a graph?



Ŧ	
2	
3	
4	

set V	:=	1, 2,	3, 4,	5,6;
set E	:=	(1,2)	(1,4)	(1,5)
		(2,3)	(2,4)	(3,4)
		(3,6)	(4,5)	(5,6);

2024/2025

47 ▶

э

Output



Solution

Output

optimal solution; objective 3								
53 simplex iterations								
у	[*]	=						
1	1							
2	1							
3	1							
4	0							
5	0							
6	0							
;								
x	[*,*]]						
:	1	2	3	4	5	6	:=	
1	0	1	0	0	0	0		
2	1	0	0	0	0	0		
3	0	1	0	0	0	0		
4	0	0	1	0	0	0		
5	1	0	0	0	0	0		
6	0	0	1	0	0	0		
;								



・ロト ・ 四ト ・ ヨト ・ ヨト

```
# V, E = some qraph, e.q.:
\mathbf{V} = [1, 2, 3, 4, 5, 6]
E = [(1,2), (1,4), (1,5), (2,3), (2,4), (3,4), (3,6), (4,5), (5,6)]
from amplpy import AMPL, Environment, DataFrame
ampl = AMPL()
ampl.option['solver'] = 'gurobi'
ampl.read("gcp.mod")
ampl.set['V'] = V
ampl.set['E'] = E
Kmax = len(V)
ampl.param['Kmax'] = Kmax
ampl.solve()
K = ampl.obj['K']
print("Colors used:", K.value())
y = ampl.var['y']
x = ampl.var['x']
for k in range(1,Kmax+1):
    if y[k].value() > .5:
        vk = [i for i in V if x[i,k].value() >.5]
        print("color {} used in {}".format(k,vk))
```
Graph coloring problem: new approach

- Previous model: minimize the number of colors used
 - determine the chromatic number K
- Different approach:
 - number of colors used is fixed
 - allow solving larger instances

Idea:

- fix number of colors
- there is no guarantee that we "color" the graph
- measure number of "bad edges" with the same color on both endpoints
- New variable: $z_{ij} = 1$ if the endpoints of edge $\{i, j\}$ have same color, 0 otherwise
- New objective: minimize the number of bad edges
 - $\bullet\,$ if the optimum is 0 \rightarrow colors assigned are feasible
 - upper bound to the chromatic number K
 - if the optimum is positive: there are bad edges
 - fixed value for the number of colors < K

$$\begin{array}{ll} \text{minimize} & \sum_{\{i,j\} \in E} z_{ij} \\ \text{subject to} & \sum_{k=1}^{K} x_{ik} = 1 & \forall i \in V \\ & x_{ik} + x_{jk} \leq 1 + z_{ij} & \forall \{i,j\} \in E; k = 1, \dots, K \\ & x_{ik} \in \{0,1\} & \forall i \in V; k = 1, \dots, K \\ & z_{ij} \in \{0,1\} & \forall \{i,j\} \in E \end{array}$$

- First constraint: exactly one color is assigned to each vertex
- Second constraint: edges {i, j} whose endpoints are assigned the same color class are bad edges → z_{ij} = 1

```
set V;
set E within {V,V};
param K;
set C := {1..K};
var x {V, C} binary;
var z {E} binary;
minimize Z: sum {(i,j) in E} z[i,j];
subject to
Color {i in V}: sum {k in C} x[i,k] = 1;
BadEdge {(i,j) in E, k in C}: x[i,k] + x[j,k] <= 1 + z[i,j];</pre>
```

イロト 不得下 イヨト イヨト

How to instantiate a graph?



Ŧ	
2	
3	
4	

set V	:=	1, 2,	3, 4,	5,6;
set E	:=	(1,2)	(1,4)	(1,5)
		(2,3)	(2,4)	(3,4)
		(3,6)	(4,5)	(5,6);

2024/2025

47 ▶

э

Out	put

	1	Z :	= 2						
	2								
	3	z	[*,*]]					
	4	:	2	3	4	5	6	:=	
	5	1	0		1	0			
	6	2		1	0				
<pre>model gcp_fixed_k.mod;</pre>	7	3		•	0		0		
data gcp.dat;	8	4		•		0			
option solver gurobi;	9	5					0		
option display_1col 0;	10	;							
solve;	11								
display y;	12	х	[*,*]]					
display x;	13	:	1	2	:	=			
	14	1	0	1					
	15	2	1	0					
	16	3	1	0					
	16 17	3 4	1 0	0 1					
	16 17 18	3 4 5	1 0 1	0 1 0					
	16 17 18 19	3 4 5 6	1 0 1 0	0 1 0 1					

- 4 ⊒ →

• • • • • • • • •

- optimum *K*:
 - smallest value such that the optimum above problems is 0
 - may be determined through binary search
- $\bullet\,\rightarrow\,{\rm check}$ improvement on CPU time used

э

Check file gcp_fixed_k.py

```
# V, E = some graph, e.g.:
V = [1, 2, 3, 4, 5, 6]
E = [(1,2), (1,4), (1,5), (2,3), (2,4), (3,4), (3,6), (4,5), (5,6)]
from amplpy import AMPL, Environment, DataFrame
ampl = AMPL()
ampl.option['solver'] = 'gurobi'
ampl.read("gcp_fixed_k.mod")
ampl.set['V'] = V
ampl.set['E'] = E
```

```
LB = 0
UB = len(V)
while UB - LB > 1:
    K = (UB + LB) // 2
    print("current K:{}\t[LB:{},UB:{}]".format(K,LB,UB))
    ampl.param['K'] = K
    ampl.solve()
    Z = ampl.obj['Z']
    print("Bad edges:", Z.value())
    if Z.value() > .5:
        LB = K
        z = ampl.var['z']
        for (i, j) in E:
            if z[i, j].value() > .5:
                print("Bad edge:", (i, j))
    else:
        UB = K
        x = ampl.var['x']
        for k in range(1, K + 1):
            vk = [i for i in V if x[i, k].value() > .5]
            print("color {} used in {}".format(k, vk))
print()
print("Chromatic number:", UB)
print("color {} used in {}".format(k, vk))
```

- Location problems:
 - k-Median
 - k-Center
 - k-Cover
 - Graph problems
 - coloring
 - Models:
 - how to formulate
 - how to improve the formulation