Data-driven Decision Making

Integer optimization, Location problems

João Pedro Pedroso

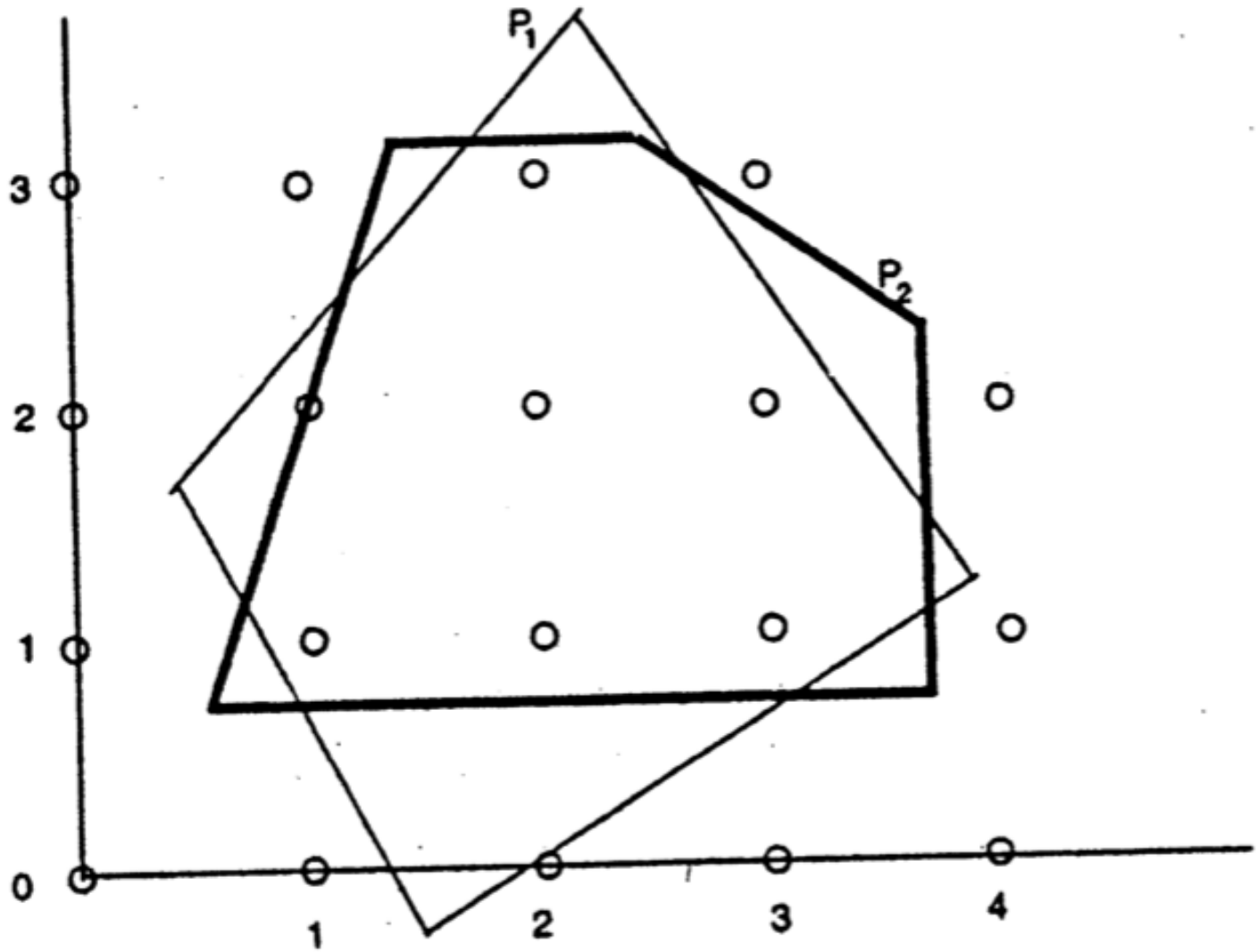
Kseniia Klimetnova

2025/2026

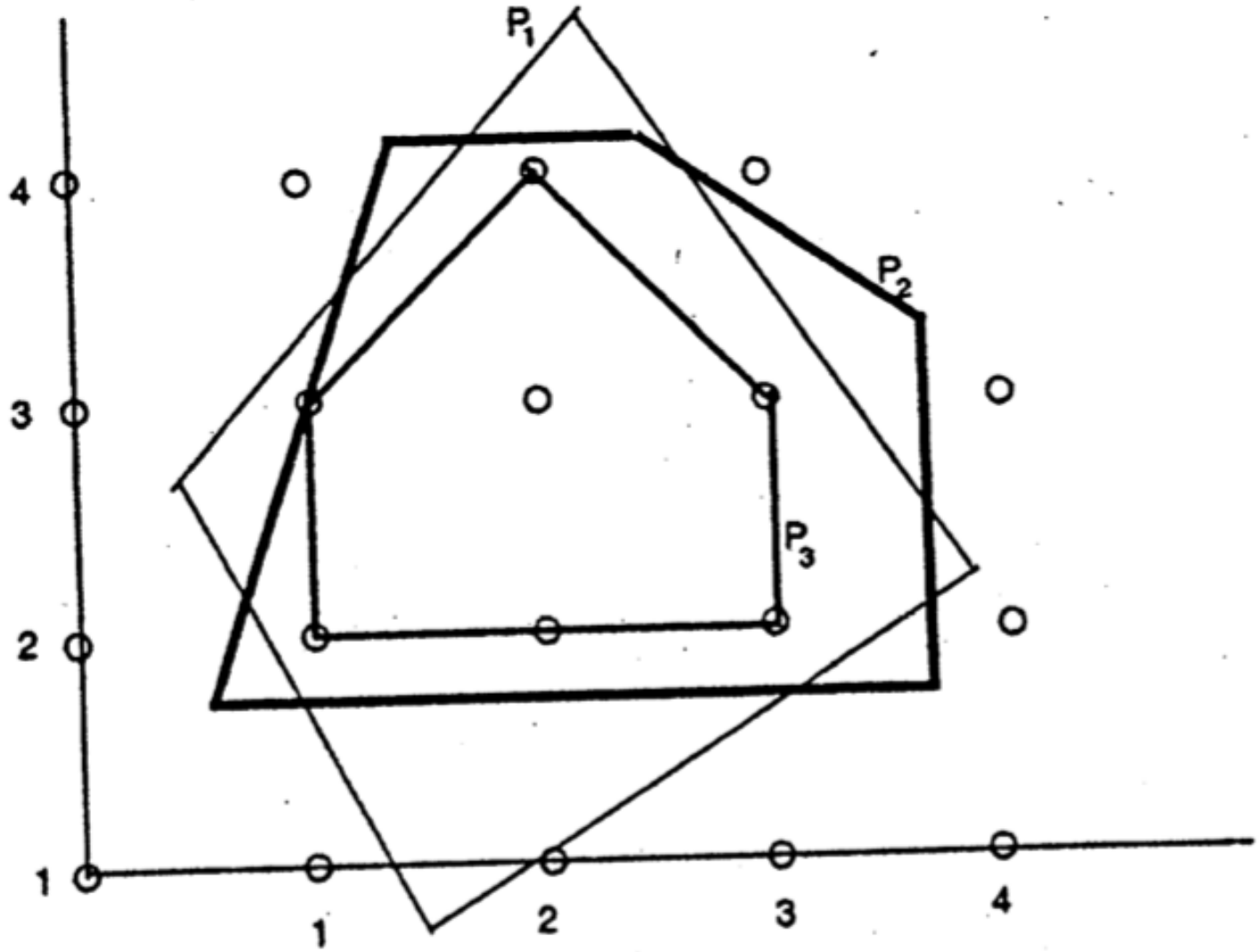
Last class: Facility Location Problems

- 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
  - no limits on total demand $\rightarrow$ big M
- Weak and strong formulations

Different formulations



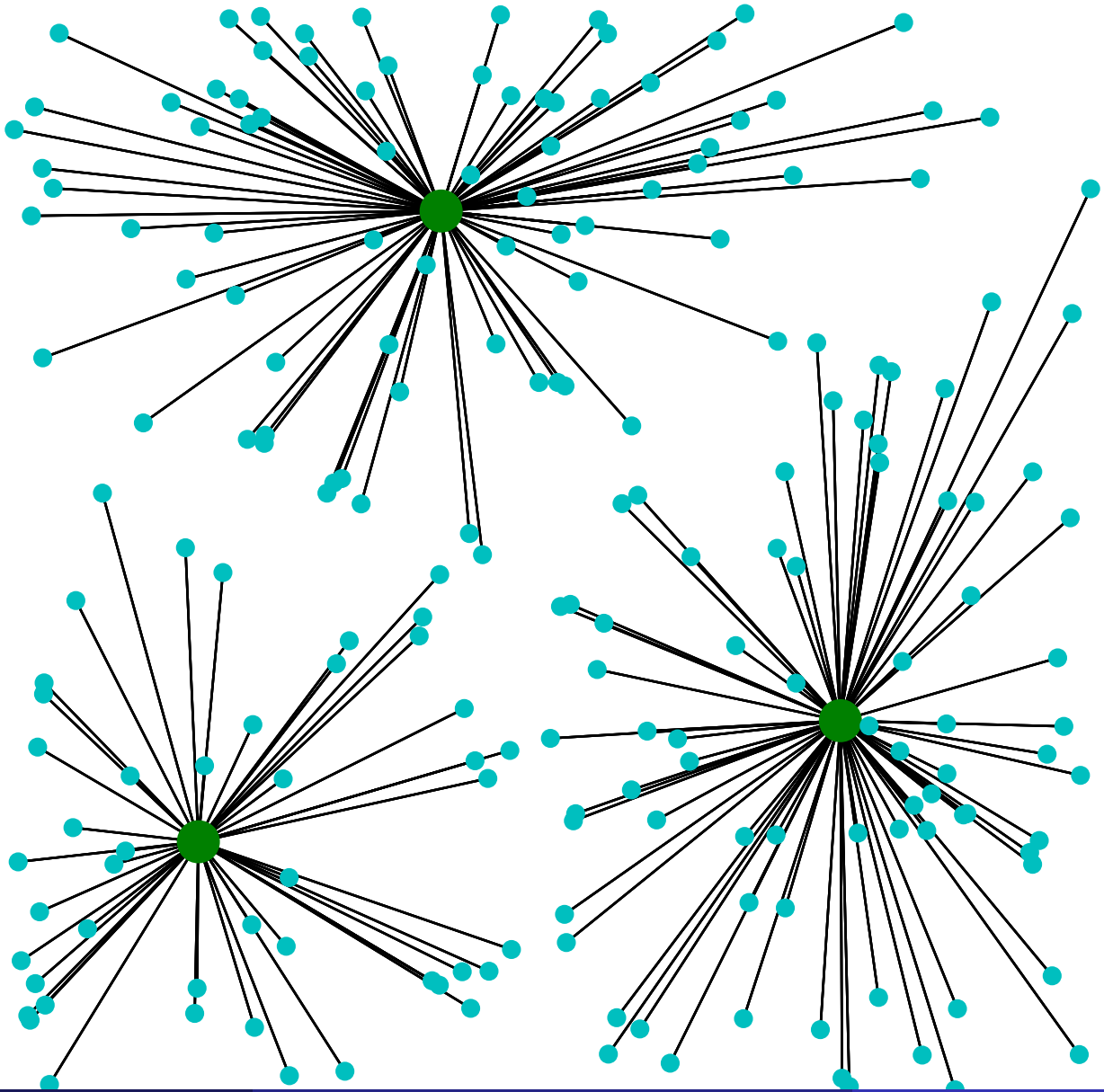
Ideal formulation



Today:

- More Facility Location Problems
  - k-Median
  - k-Center
  - k-Cover
- Graph problems

The k-Median problem



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 $j \rightarrow c_{ij}$
  - set of customers $\rightarrow \{1, \ldots, n\}$
  - set of potential places for facilities $\rightarrow \{1, \ldots, m\}$
  - commonly, facilities and customers share the same set of points
- Variables:

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

$$y_j = \begin{cases} 1 & \text{when facility } j \text{ is open} \\ 0 & \text{otherwise} \end{cases}$$

The k-Median problem

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

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

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

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

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

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

- 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

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

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

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

  $\rightarrow$ lead to worse values in the linear relaxation

AMPL model

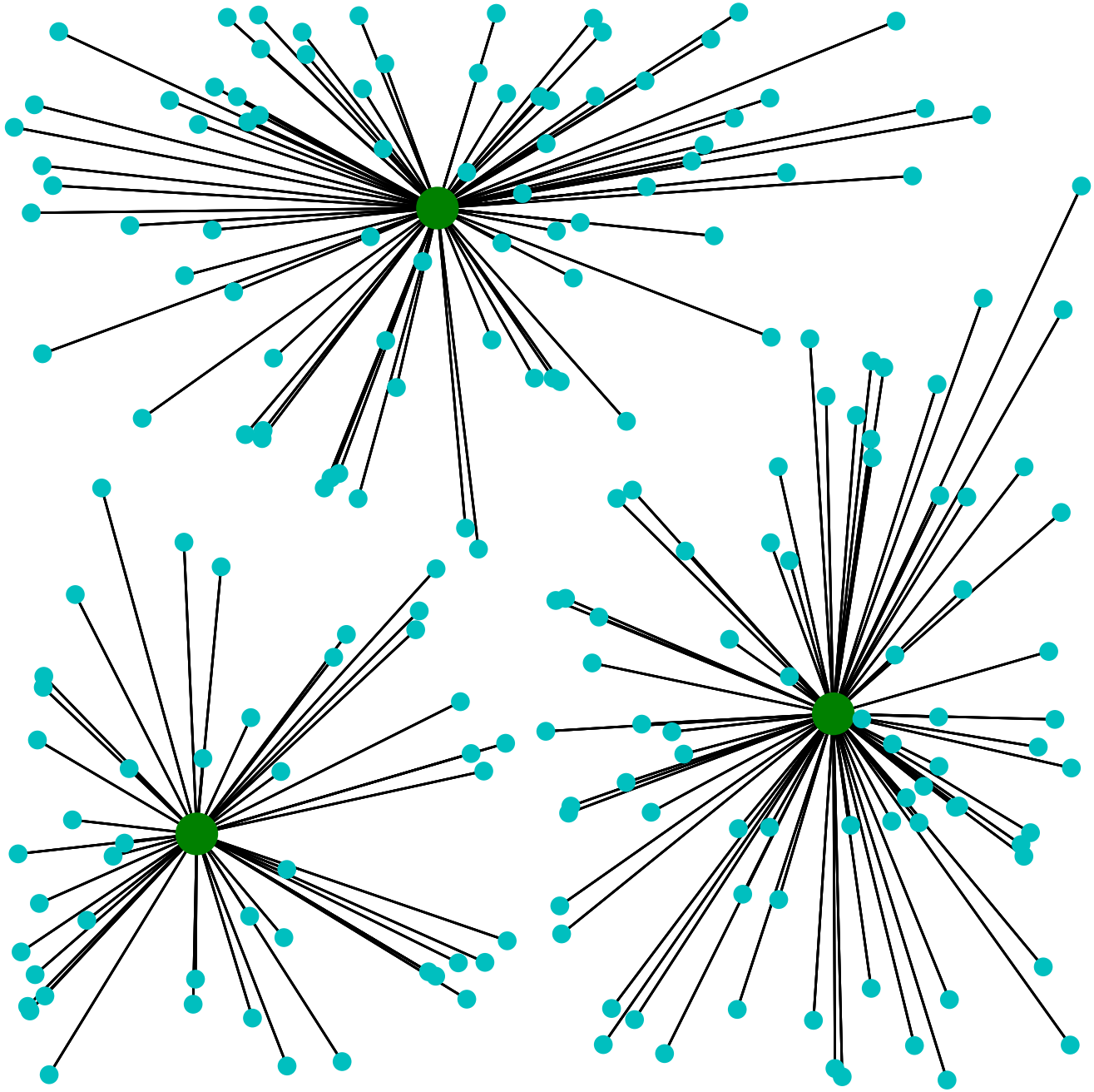
```
param n;  # number or customers
param m;  # number or facilities
param c {1..n, 1..m};
param k;

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];
```

AMPL data

```
data;
param n := 5;
param m := 3;
param k := 2;
param c (tr) :    # (tr) --> transposed
             1  2  3  4  5 :=
      1      4  5  6  8  10
      2      6  4  3  5  8
      3      9  7  4  3  4 ;
```

Illustration

- 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



Programming

- AMPL model: see above
- AMPL data: replaced by a Python program

→ see file `kmedian_plt.py`

The k-Center Problem

The k-Center Problem

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 $\rightarrow \{1, \ldots, n\}$
  - set of potential places for facilities $\rightarrow \{1, \ldots, m\}$
  - commonly, facilities and customers share the same set of points
- Variables (same as k-Median):

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

$$y_j = \begin{cases} 1 & \text{when facility } j \text{ is open} \\ 0 & \text{otherwise} \end{cases}$$

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

The k-Center problem

$$
\begin{aligned}
\text{minimize} \quad & z \\
\text{subject to:} \quad & \sum_{j=1}^{m} x_{ij} = 1 && \text{for } i = 1, \cdots, n \\
& \sum_{j=1}^{m} y_j = k \\
& x_{ij} \leq y_j && \text{for } i = 1, \cdots, n; j = 1, \cdots, m \\
& \sum_{j=1}^{m} c_{ij} x_{ij} \leq z && \text{for } i = 1, \cdots, n \\
& x_{ij} \in \{0, 1\} && \text{for } i = 1, \cdots, n; j = 1, \cdots, m \\
& y_j \in \{0, 1\} && \text{for } j = 1, \cdots, m
\end{aligned}
$$

- 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:

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

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

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

AMPL model

```
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 >= 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;
```

Programming: same as with kmedian

- Observe difference in performance
- Observe difference in the solution

Techniques in linear optimization

- "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:

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

Related topic: minimization of the absolute value

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 $z \geq x$ and $z \geq -x$
  - z replaces $|x|$ in the objective function

Modeling tip

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

- In integer optimization $\rightarrow$ 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

The k-Cover Problem

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

The k-Cover Problem

- 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 $S \subseteq V$ is called a cover if every vertex $i \in V$ is adjacent to at least one of the vertices in S
- Idea: optimum value of the k-center problem $\leq \theta$ if there exists a cover with cardinality $|S| = k$ on graph G_θ.

Notation

- 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_θ
  - $a_{ij} = 1$ if vertices i and j are adjacent, 0 otherwise
- We need to determine whether or not graph G_θ has a cover $|S| = k$
- We can do that by solving integer-optimization model $\rightarrow$ k-cover problem on G_θ

The k-Cover Problem: model

$$\text{minimize} \quad \sum_{i=1}^{n} z_i$$

$$\text{subject to} \quad \sum_{j=1}^{m} a_{ij} y_j + z_i \geq 1 \qquad \text{for } i = 1, \cdots, n$$

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

$$z_i \in \{0, 1\} \qquad \text{for } i = 1, \cdots, n$$

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

- 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

Binary search

- 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
    - $\rightarrow$ reduce θ
  - 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

AMPL model

```
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;
```

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] for (i,j) in c)
```

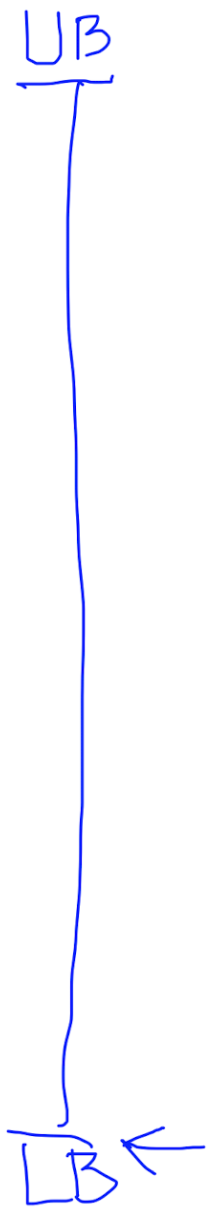
Programming: binary search

```
delta = 1.e-4  # tolerance
LB = 0
UB = max(c[i,j] for (i,j) 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
```

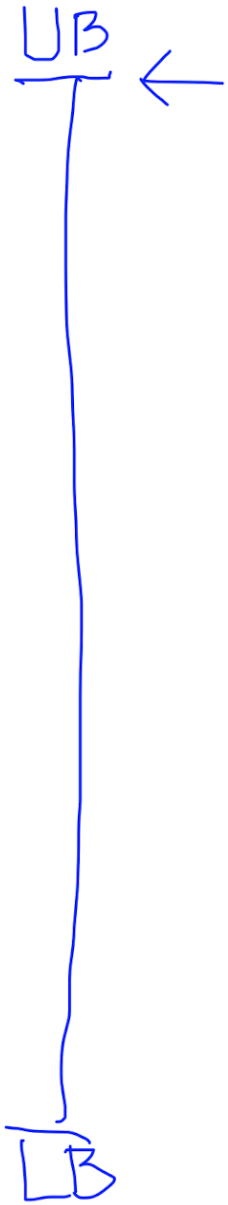
Remarks

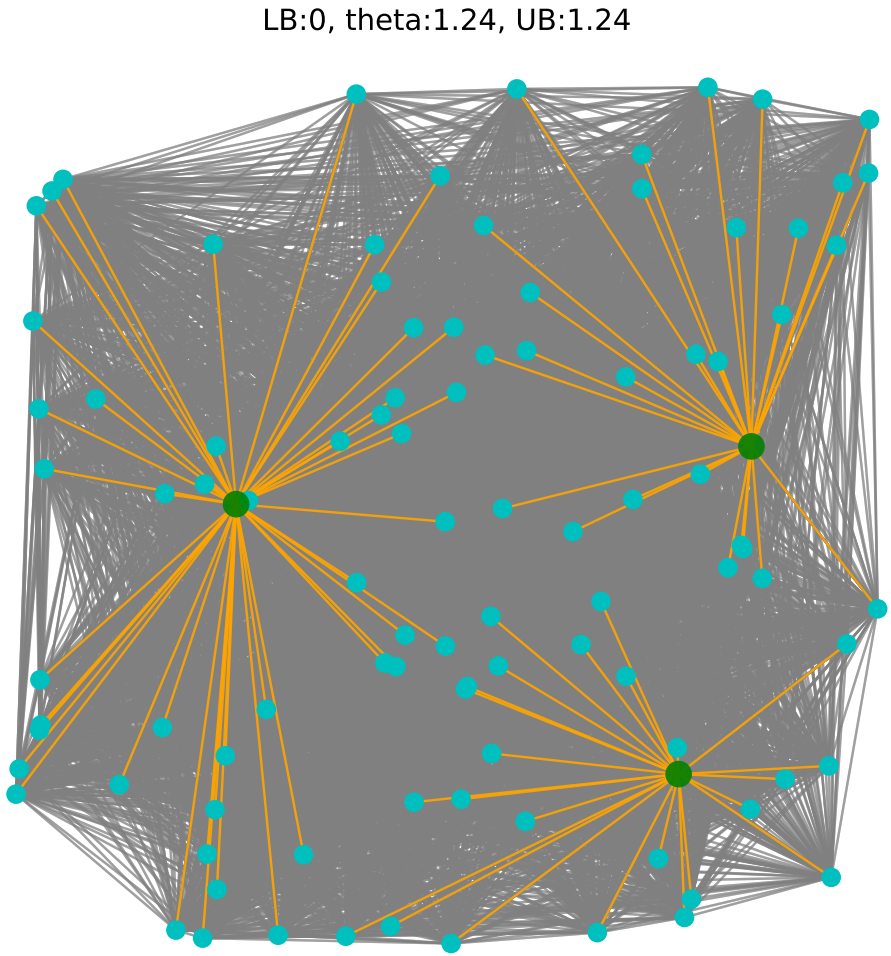
- k-cover within binary search $\rightarrow$ time comparable k-median check
- in practice, k-center solution is usually preferable to k-median
  - longest time required for serving a customer
  - may be large on the k-median solution.

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

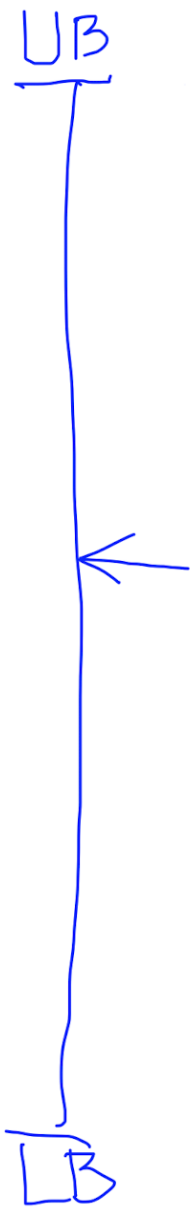


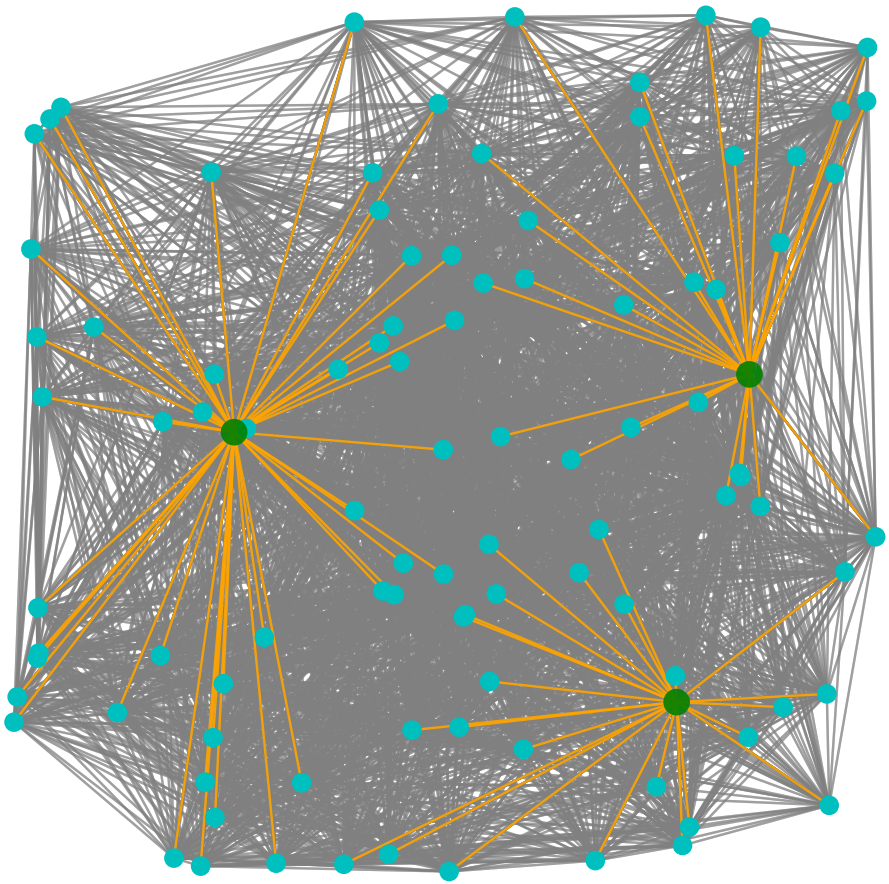
Using k-Cover and binary search: $\theta = \max c_{ij}$ (redundant)



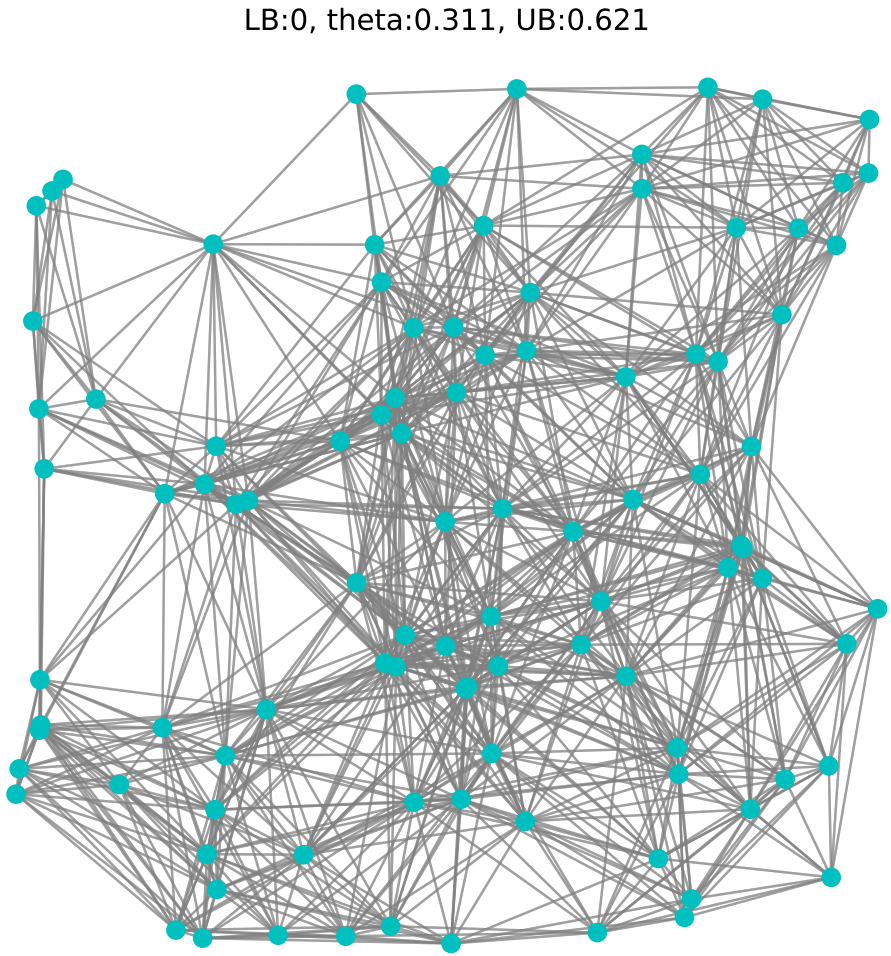


Using k-Cover and binary search: $\theta = 0.621$

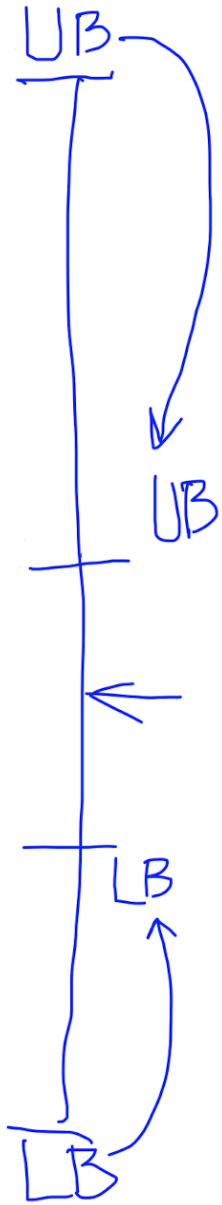


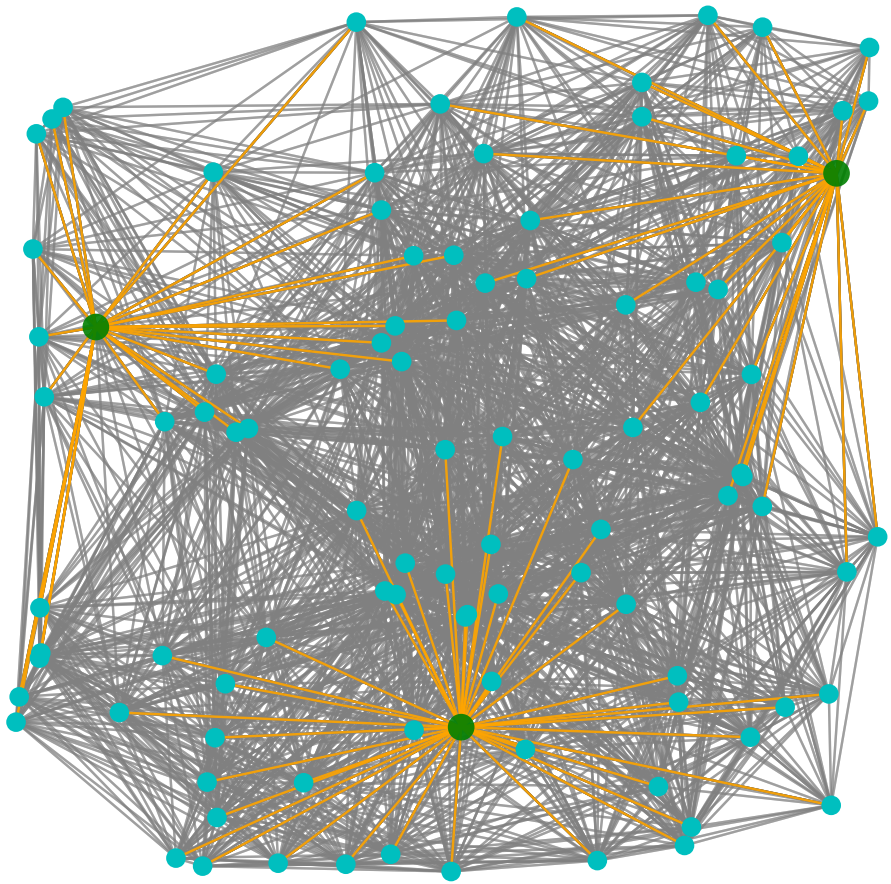


Using k-Cover and binary search $\theta = 0.311$

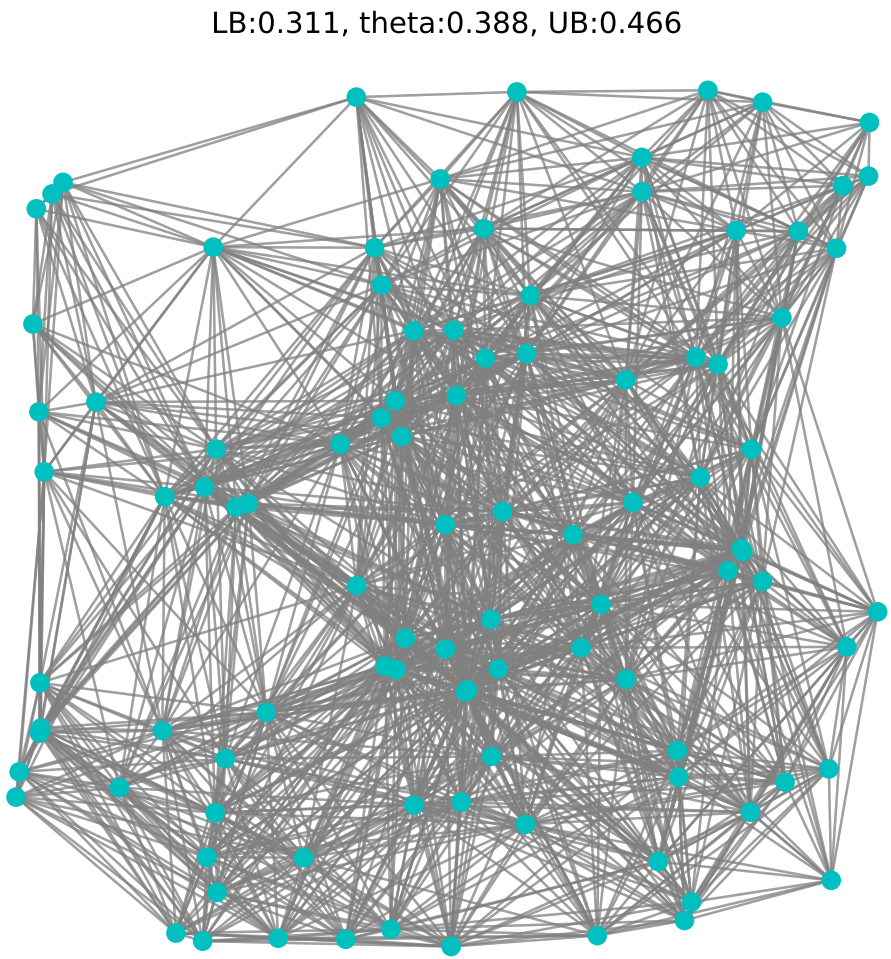


Using k-Cover and binary search $\theta = 0.466$

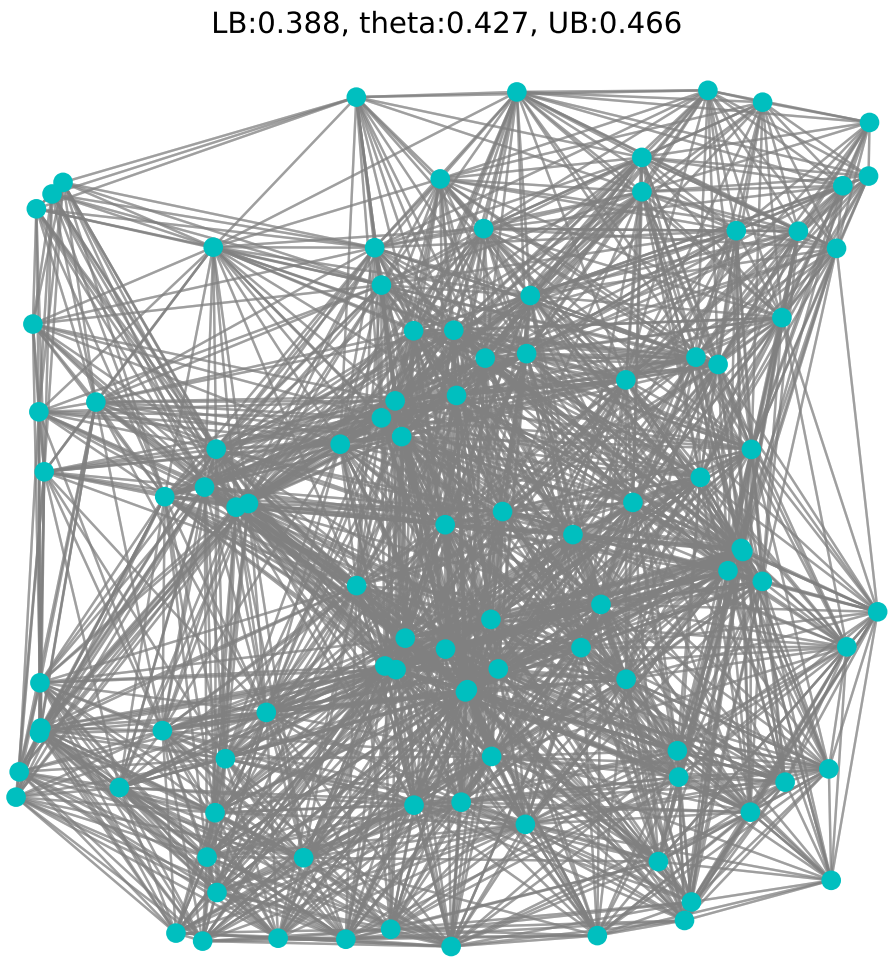




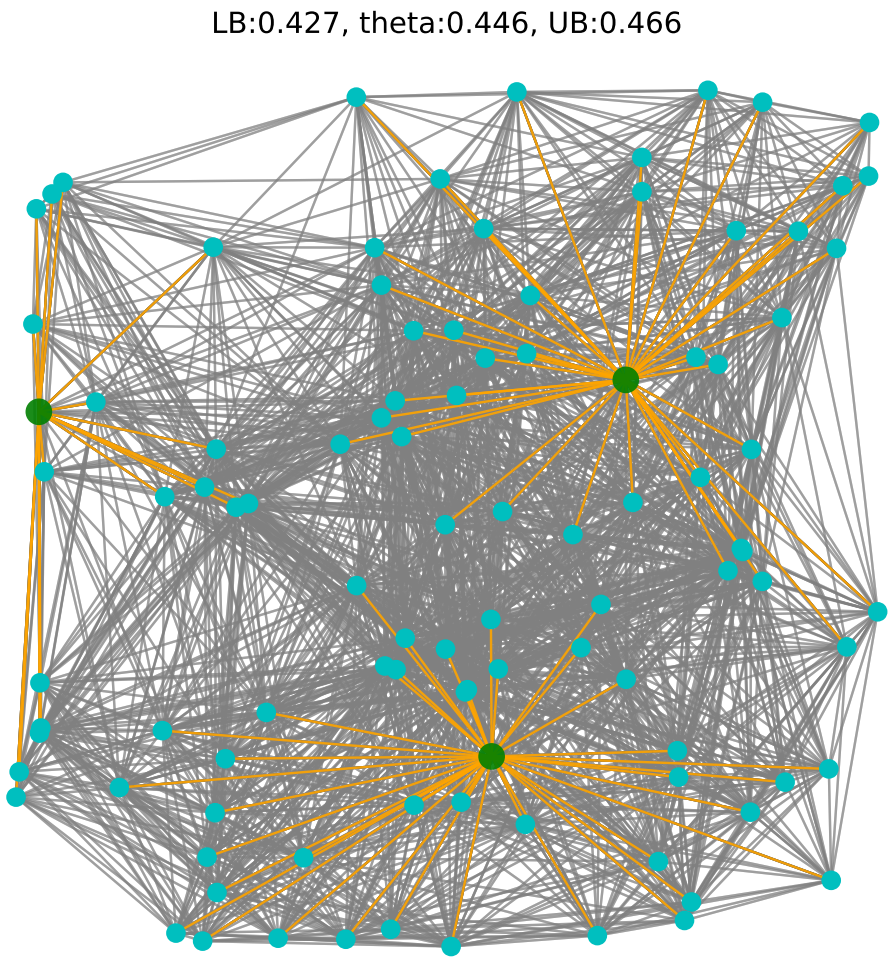
Using k-Cover and binary search $\theta = 0.388$



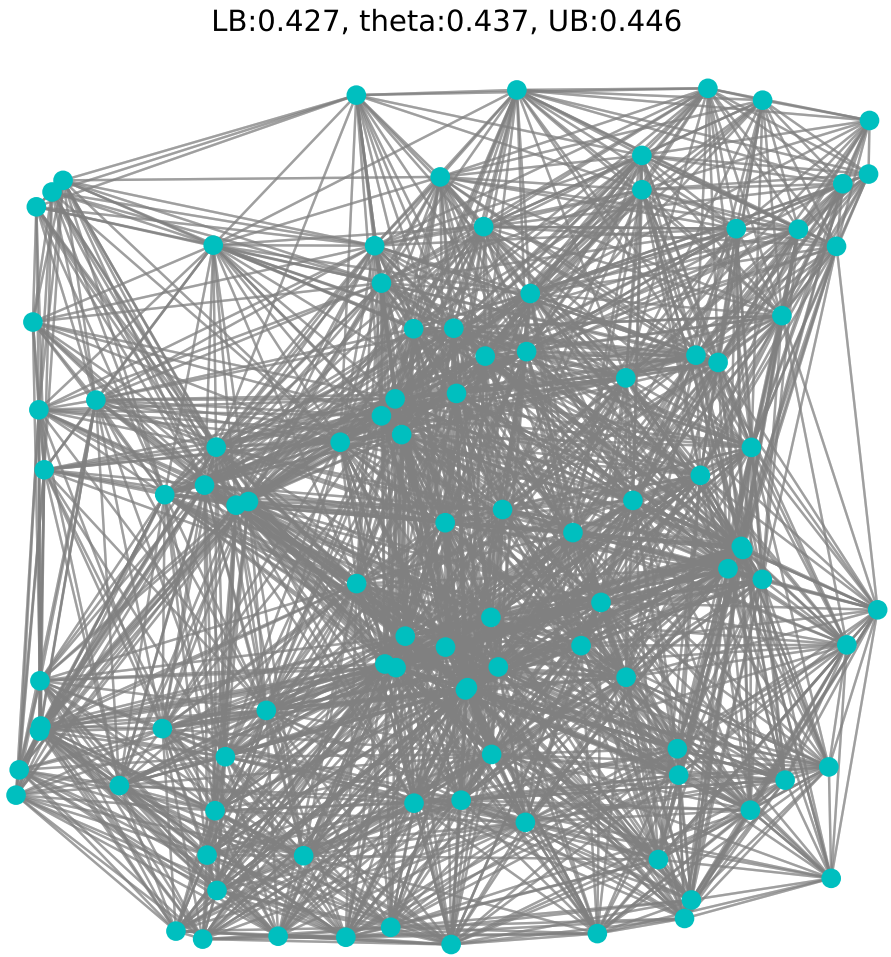
Using k-Cover and binary search $\theta = 0.427$



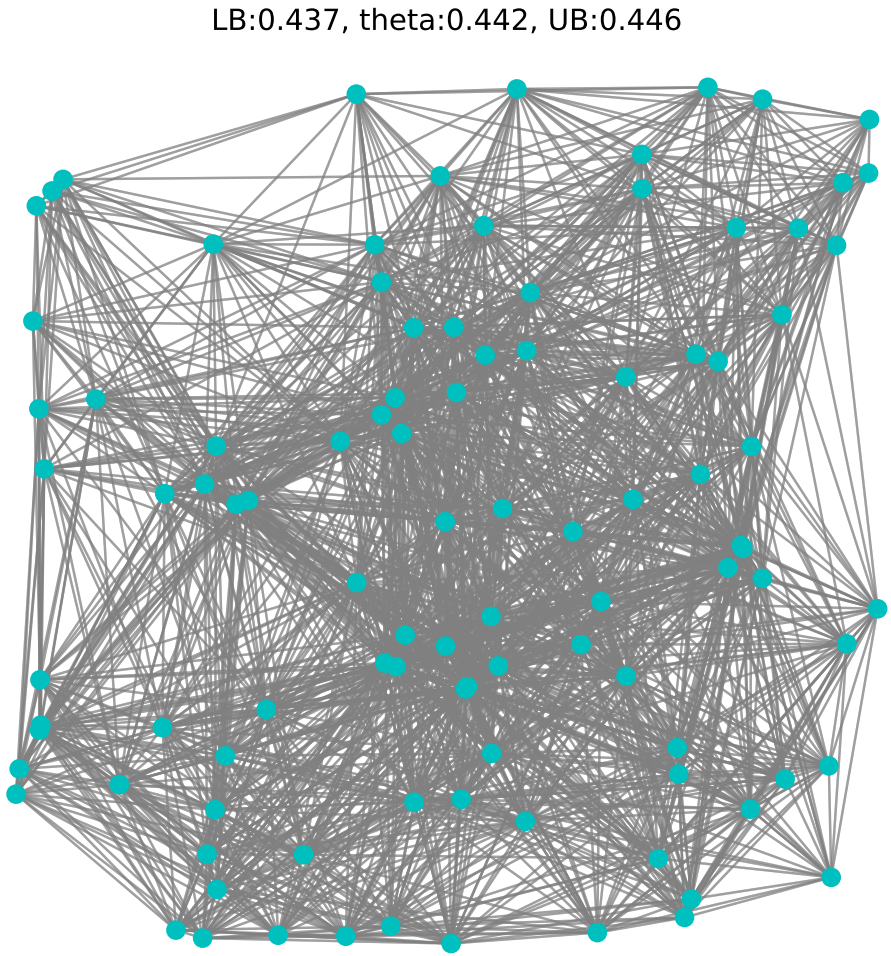
Using k-Cover and binary search $\theta = 0.446$



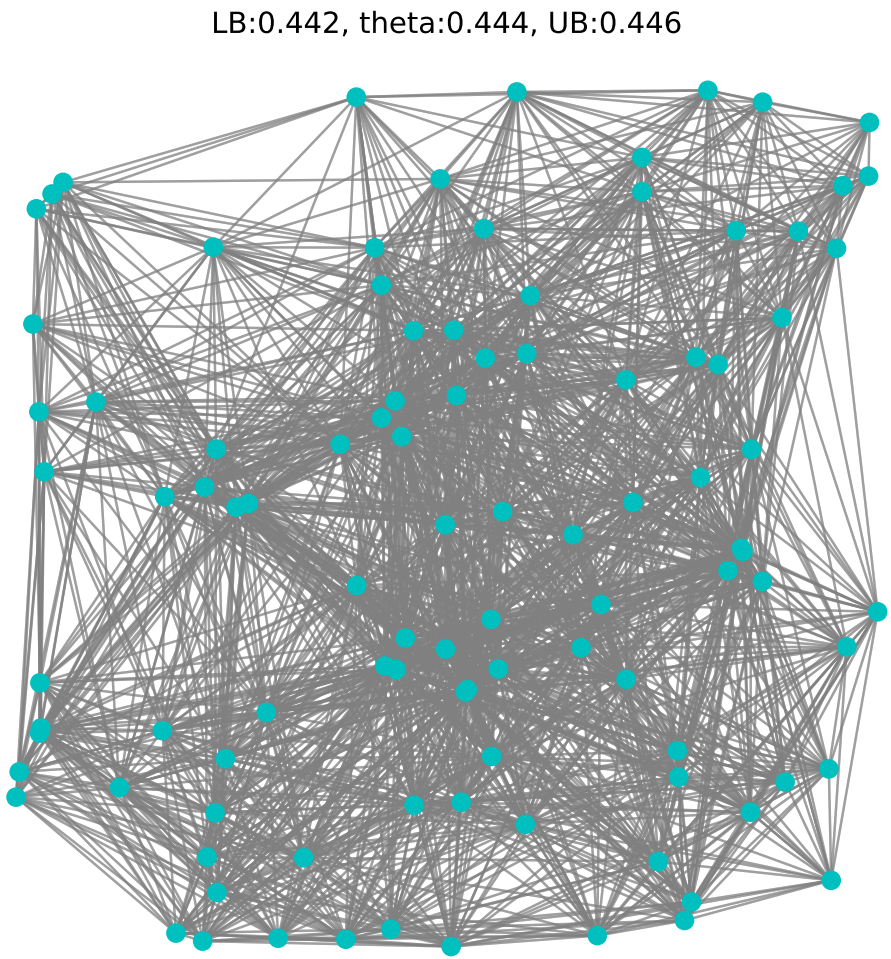
Using k-Cover and binary search $\theta = 0.437$



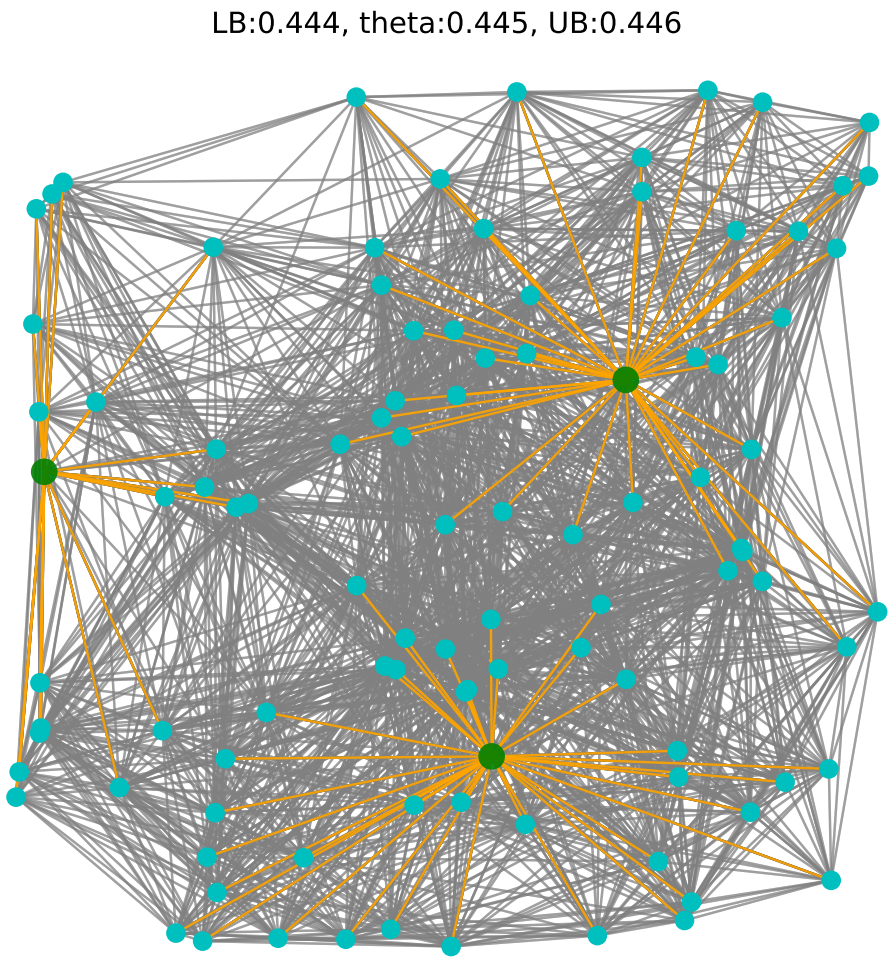
Using k-Cover and binary search $\theta = 0.442$



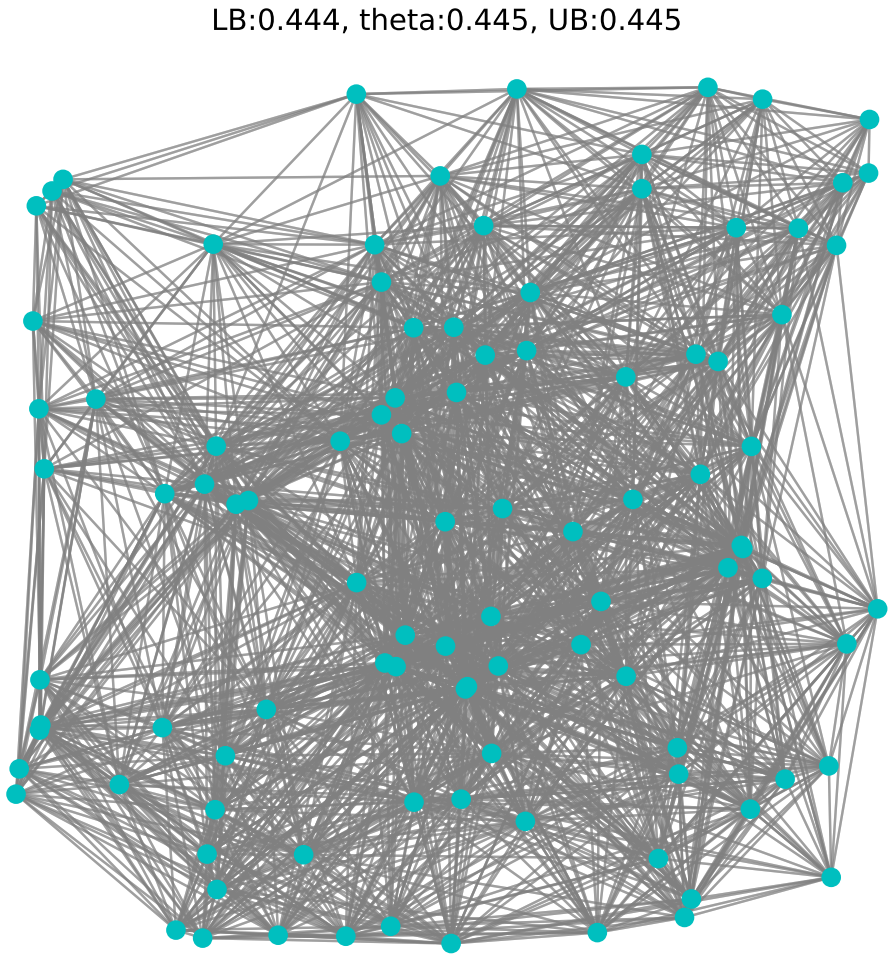
Using k-Cover and binary search $\theta = 0.444$



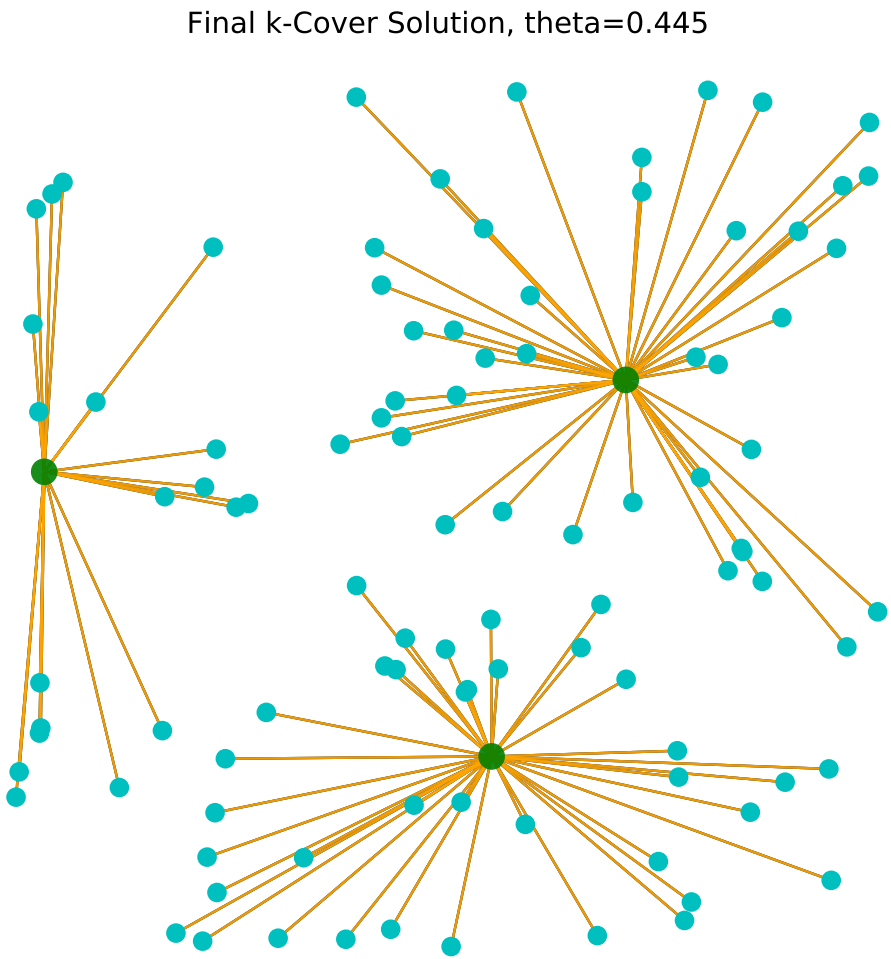
Using k-Cover and binary search $\theta = 0.445$



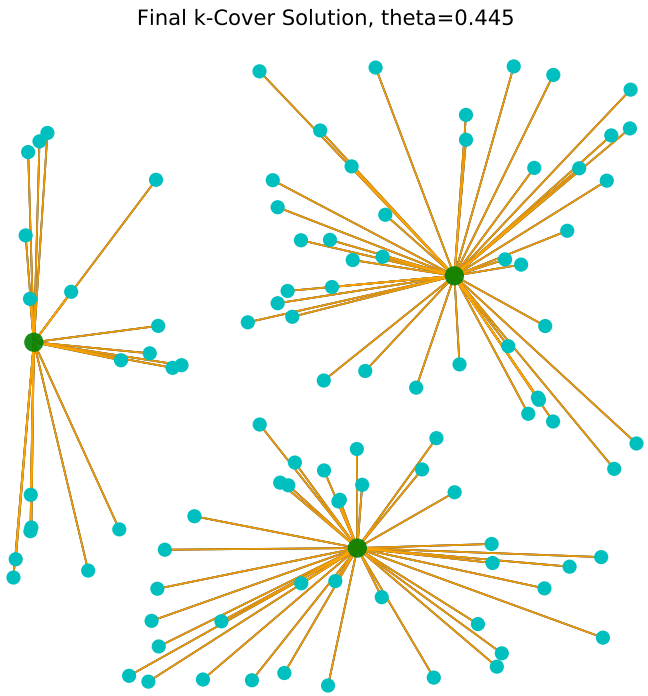
Using k-Cover and binary search $\theta = 0.445-$

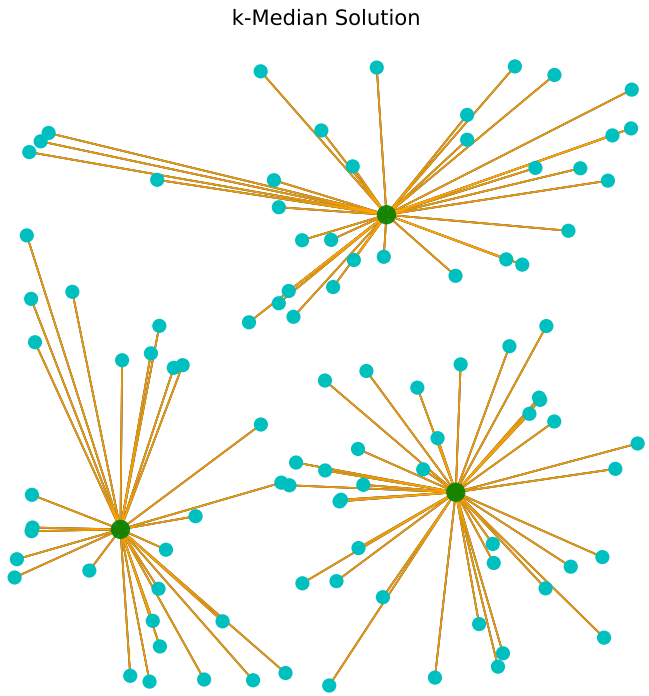


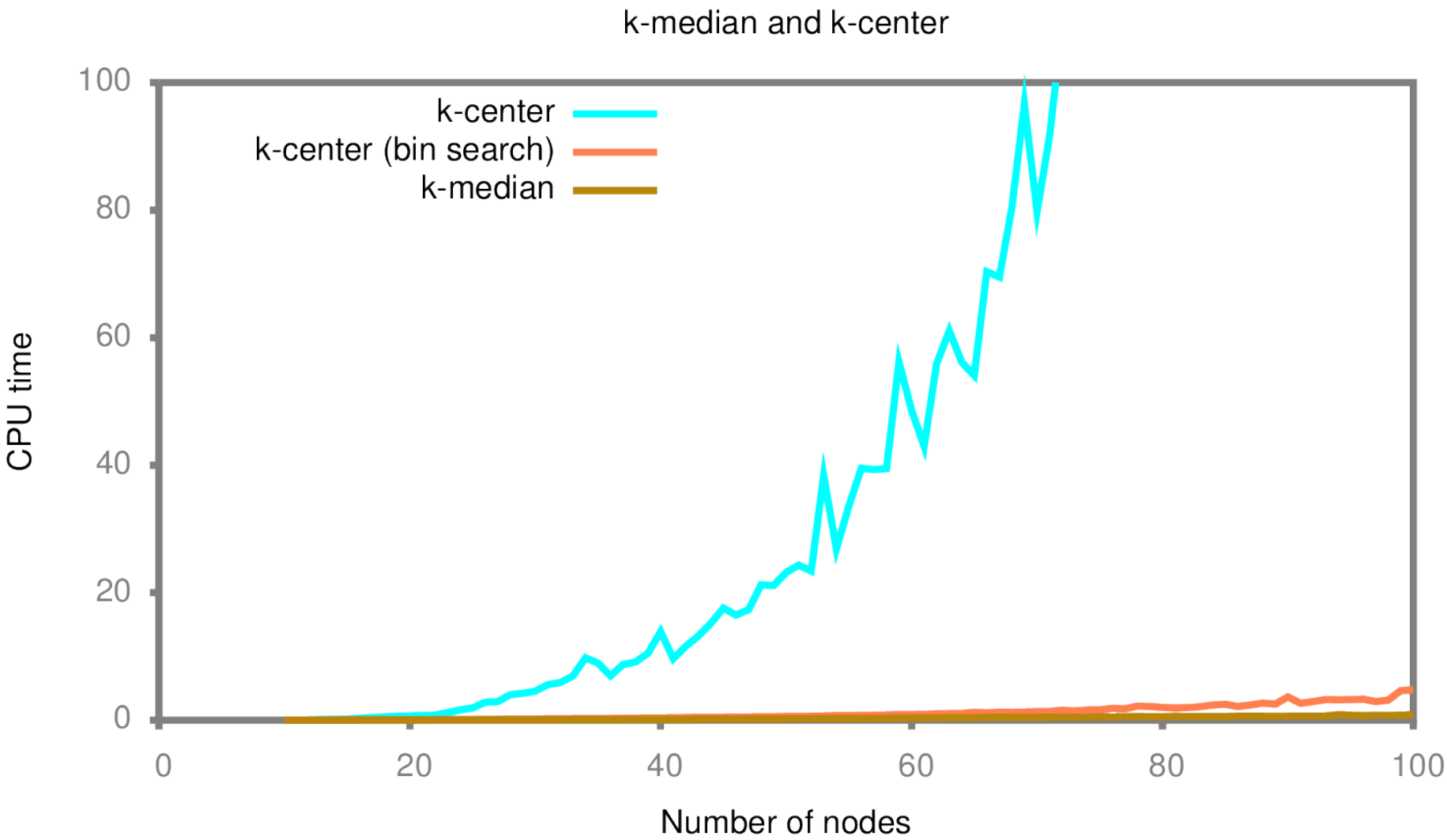
Using k-Cover and binary search $\theta = 0.445+$ (final)



Using k-Cover and k-Center






k-median and k-center
k-center
k-center (bin search)
k-median
CPU time
Number of nodes
0
20
40
60
80
100

Graph problems

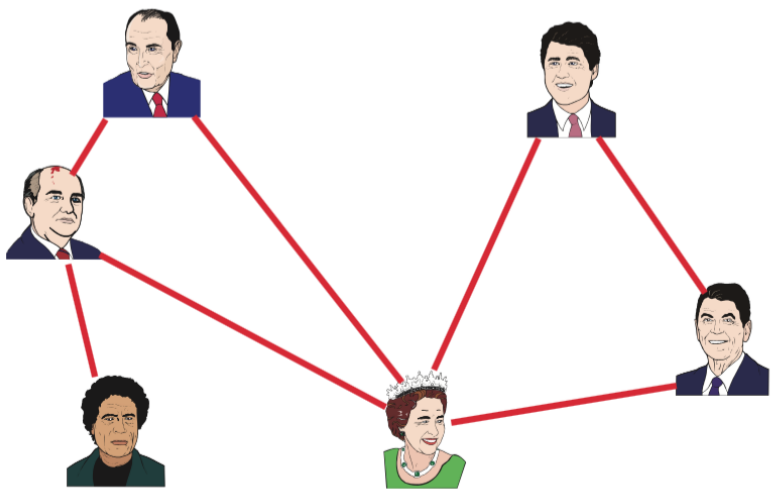
Graph problems

This class:

- Based on the definition of a graph
  - abstract object composed of vertices and edges
    - edge $\rightarrow$ link between two 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
  - 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
  - in graph theory $\rightarrow$ called an edge, arc or line
- This representation is a graph $\rightarrow$ friendship scenario becomes very easy to grasp



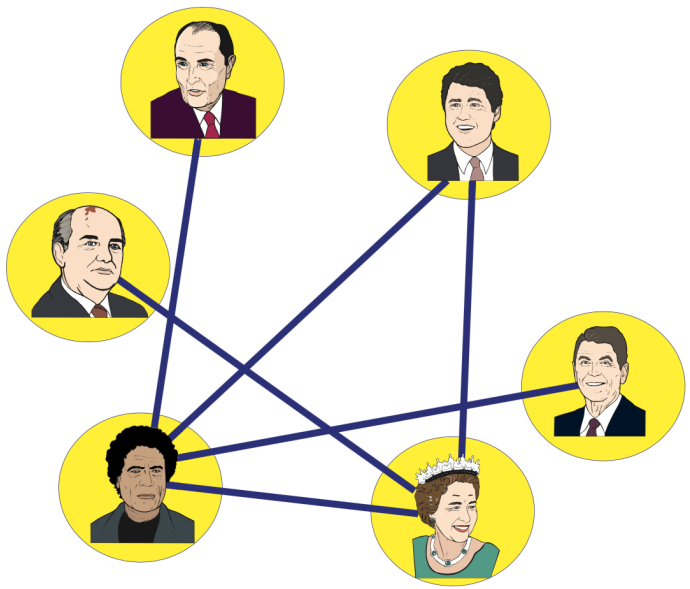
Some terminology

- Undirected graphs: lines connecting two vertices have no implied direction
  - lines with no direction $\rightarrow$ edges
- Directed graphs: direction of lines connecting two vertices is important
  - directed lines $\rightarrow$ 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.

Graph coloring problem

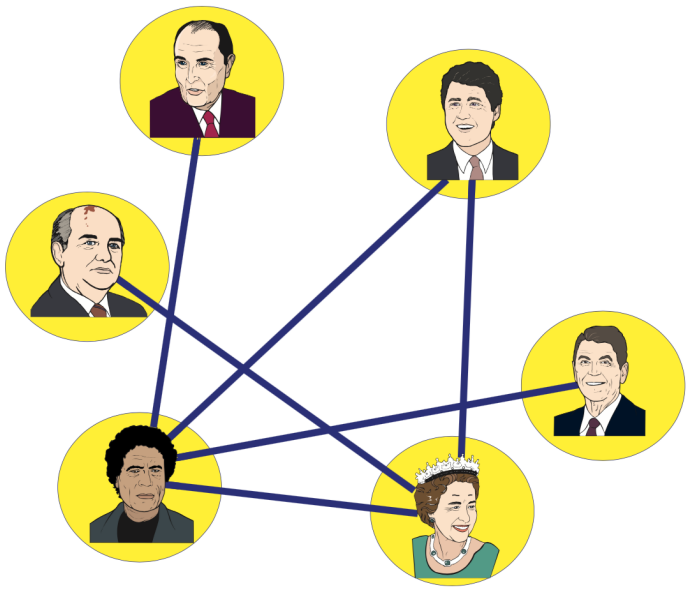
Graph coloring problem

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?



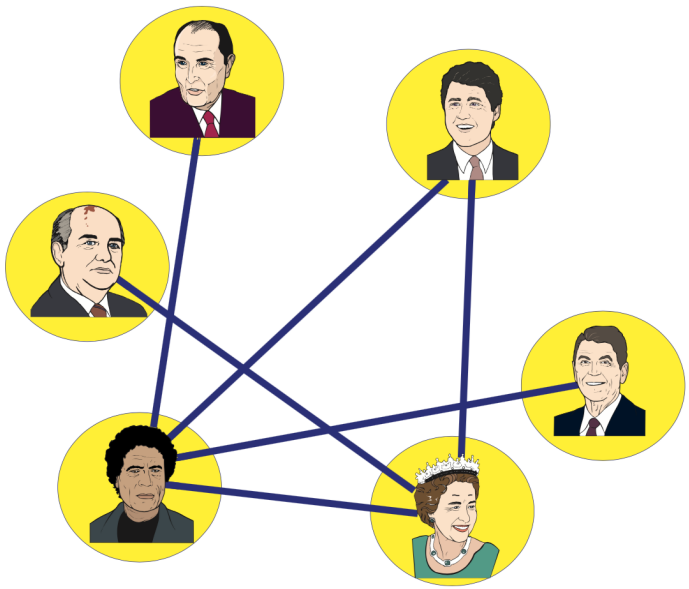
Graph coloring problem

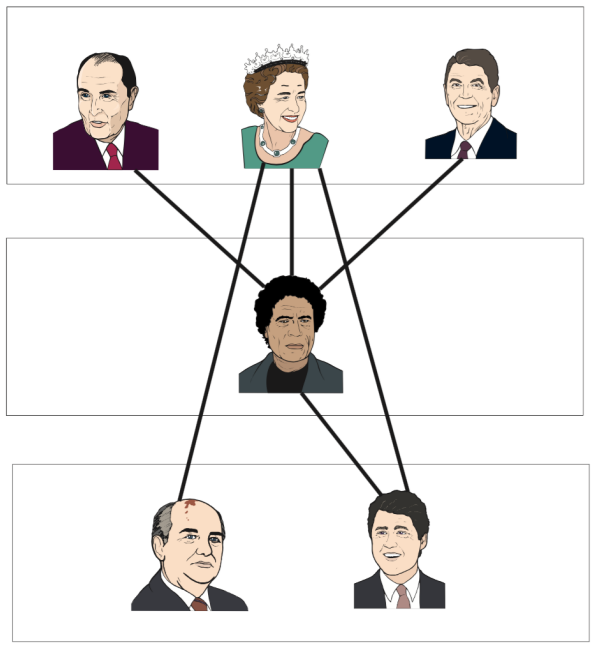
- Graph:
  - nodes $\rightarrow$ persons
  - edge between two persons $\rightarrow$ not good friends
  - three classes are enough to separate persons on unfriendly terms



Graph coloring problem

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

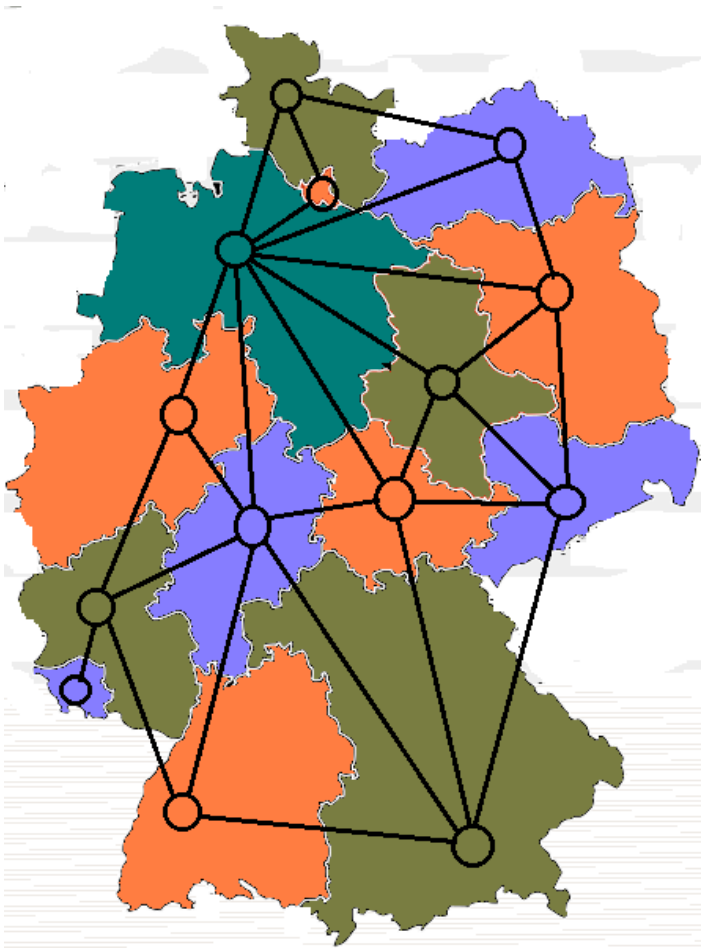




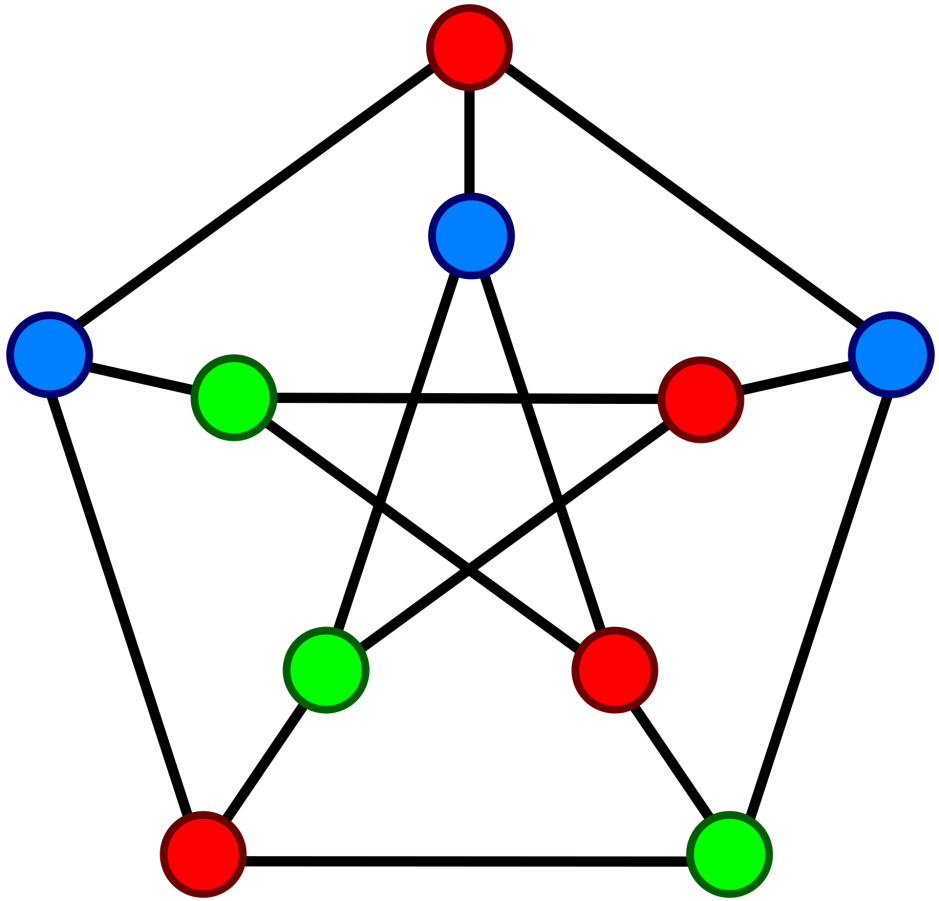
Graph coloring: applications

1. Map Coloring
   - maps of countries
   - adjacent coutries cannot have the same color
   - four colors are sufficient ("Four Color Theorem")
2. Scheduling and timetabling
   - different courses
   - students enrolled in courses
   - how to schedule exams so that no common students are scheduled at same time?
     - vertex $\rightarrow$ course
     - edge $\rightarrow$ some student on both courses
3. Mobile Radio Frequency Assignment
   - frequencies are assigned to towers
   - if towers are "close enough" $\rightarrow$ interference $\rightarrow$ frequencies must be different
     - tower $\rightarrow$ vertex
     - two towers are "close" $\rightarrow$ edge between them

Graph coloring: illustration



Graph coloring: illustration



Graph coloring problem

- 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
  - $V_i \cap V_j = \emptyset, \forall i \neq j \rightarrow$ *no overlap*
  - $\bigcup_{j=1}^{K} V_j = V \rightarrow$ *union of subsets = full set of vertices*
- Each $V_i (i = 1, 2, \ldots K)$ 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 $\rightarrow$ *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, ...

Graph coloring problem: notation

- Requirement: upper bound $K_{\max}$ of the number of colors
- Optimal number of colors K: integer $1 \leq K \leq K_{\max}$
- 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 $y_k = 1 \rightarrow$ 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

$$\text{minimize} \quad \sum_{k=1}^{K_{\max}} y_k$$

$$\text{subject to} \quad \sum_{k=1}^{K_{\max}} x_{ik} = 1 \qquad \forall i \in V$$

$$x_{ik} + x_{jk} \leq y_k \qquad \forall \{i,j\} \in E; k = 1, \ldots, K_{\max}$$

$$x_{ik} \in \{0,1\} \qquad \forall i \in V; k = 1, \ldots, K_{\max}$$

$$y_k \in \{0,1\} \qquad k = 1, \ldots, K_{\max}$$

- 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

Remarks

- Most mathematical optimization solvers use branch-and-bound
- 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 $V_1 = 1, 2, 3, V2 = 4, 5$ and $V_1 = 4, 5, V2 = 1, 2, 3$ 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
  - $\rightarrow$ use preferentially color classes with low subscripts
  - may considerably improve solving time

$$y_k \geq y_{k+1} \quad k = 1, \ldots, K_{\max} - 1$$

Modeling tip 5

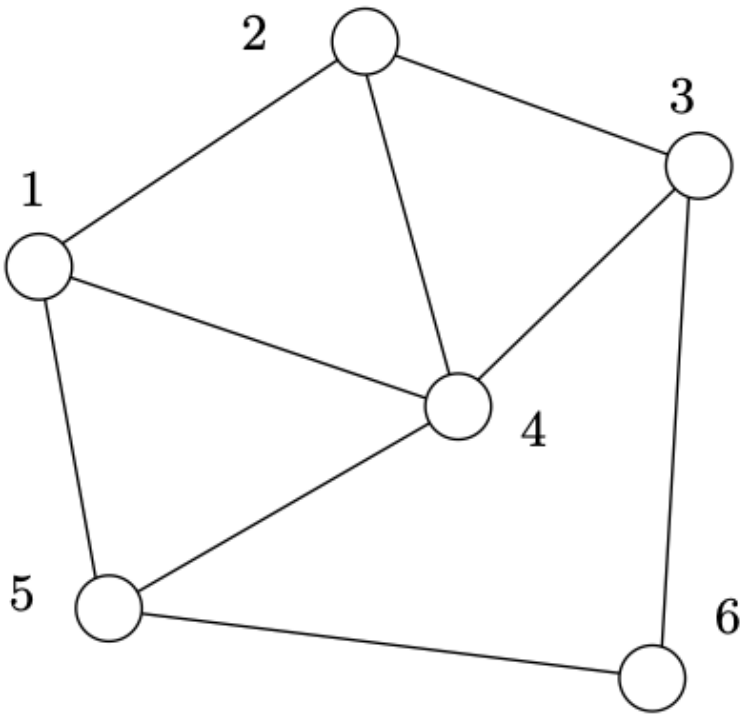
When there is symmetry in a formulation, add constraints for removing it.

- 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

AMPL model

```
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] <= y[k];
NoSym {k in 1..Kmax-1}: y[k] >= y[k+1];
```

How to instantiate a graph?




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

How to solve it?

```
model gcp.mod;
data gcp.dat;
option solver gurobi;
option display_1col 0;
solve;
display y;
display x;
```

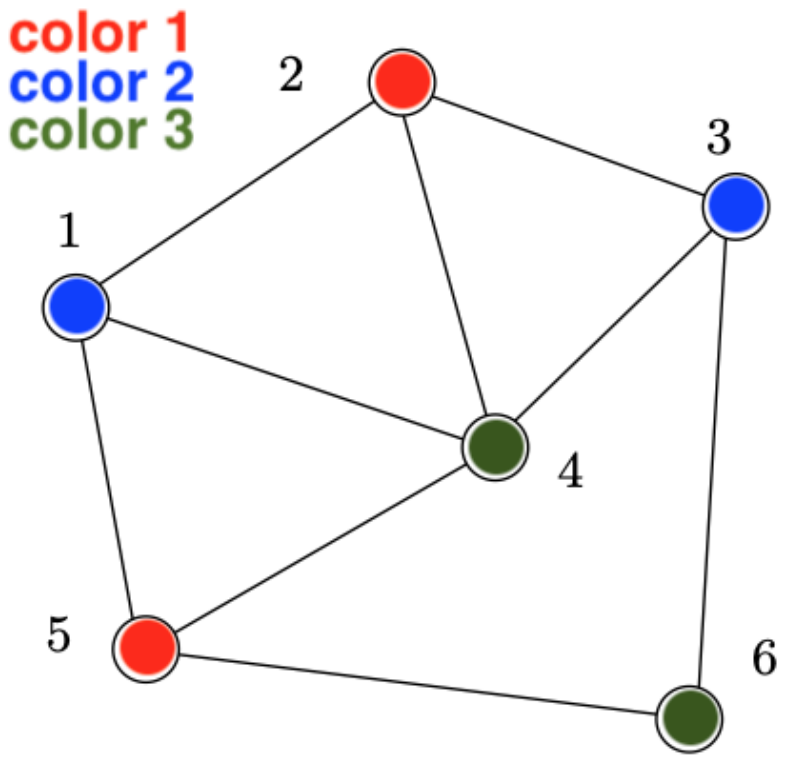
Output

```
optimal solution; objective 3
53 simplex iterations
y [*] :=
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
;
```

Solution

Output

```
optimal solution; objective 3
53 simplex iterations
y [*] :=
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
;
```



Python (check file gcp.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.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
  - 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 $< K$

Graph coloring problem: minimize *bad edges*

$$\text{minimize} \quad \sum_{\{i,j\} \in E} z_{ij}$$

$$\text{subject to} \quad \sum_{k=1}^{K} x_{ik} = 1 \qquad \forall i \in V$$

$$x_{ik} + x_{jk} \leq 1 + z_{ij} \qquad \forall \{i,j\} \in E; k = 1, \ldots, K$$

$$x_{ik} \in \{0,1\} \qquad \forall i \in V; k = 1, \ldots, K$$

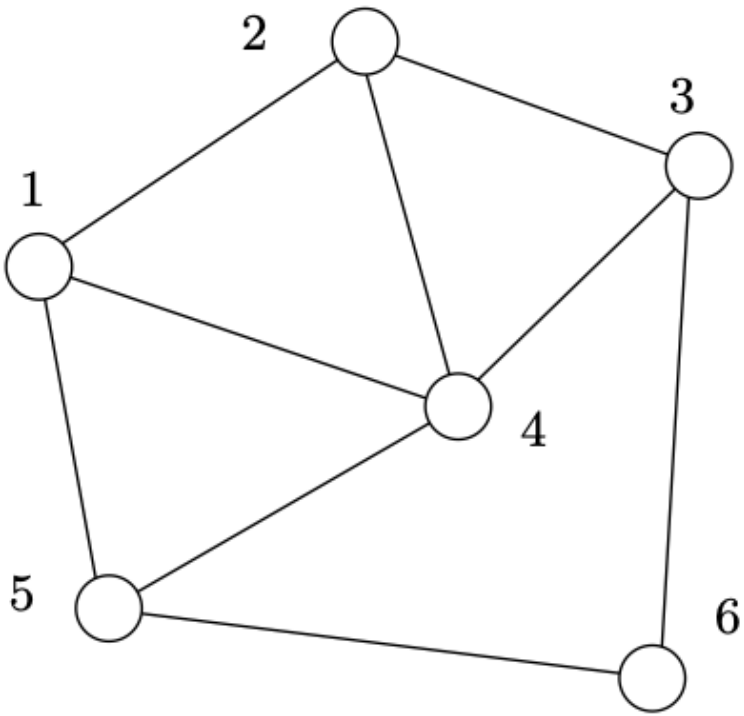
$$z_{ij} \in \{0,1\} \qquad \forall \{i,j\} \in E$$

- 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 $\rightarrow z_{ij} = 1$

AMPL model

```
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];
```

How to instantiate a graph?




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

How to solve it?

```
model gcp_fixed_k.mod;
data gcp.dat;
option solver gurobi;
option display_1col 0;
solve;
display y;
display x;
```

Output

```
Z = 2

z [*,*]
:   2   3   4   5   6    :=
1   0   .   1   0   .
2   .   1   0   .   .
3   .   .   0   .   0
4   .   .   .   0   .
5   .   .   .   .   0
;

x [*,*]
:   1   2    :=
1   0   1
2   1   0
3   1   0
4   0   1
5   1   0
6   0   1
;
```

Remarks

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

Binary search method for solving the graph coloring problem.

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))
```

This lesson

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