Optimization with Gurobi and Python

João Pedro PEDROSO

INESC Porto and Universidade do Porto, Portugal
jpp@fc.up.pt

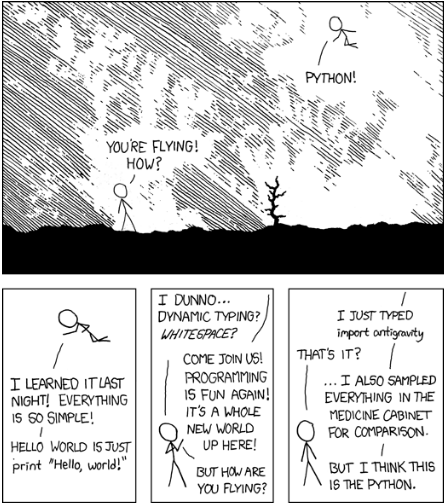
Workshop at Universidade dos Açores
September 2011

Gurobi – a one-page explanation

- Optimization system by Z. **Gu**, E. **Ro**thberg, and R. **Bi**xby

- Very high performance, cutting-edge solvers:
  - linear programming
  - quadratic programming
  - mixed-integer programming
- Advanced presolve methods
- MILP and MIQP models:
  - cutting planes
  - powerful solution heuristics
- Free academic license

Why Python?



- Everything can be done after loading a module!
- Optimization allowed:

  `import gurobipy`
- Use algorithms on graphs:

  `import networkX`
  `import matplotlib`
- Allows levitation:

  `import antigravity (?)`

Python — a one-page explanation

- Simple types: bools, integers, floats, strings (*immutable*)
- Complex types:
  - lists: sequences of elements (of any type; *mutable*)
    - indexed by an integer, from 0 to size-1
    - `A=[1,5,3,7], A.append(5), a=A.pop(), a=A[3], A[4]="abc", A.remove(6), A.sort()`
  - tuples: as lists, but *immutable* $\rightarrow$ may be used as indices
    - `T=(1,5,3,7), t=T[3]`
  - dictionaries: mappings composed of pairs *key, value* (*mutable*)
    - indexed by an integer, from 0 to size-1
    - `D = {}, D[872]=6, D["pi"]=3.14159, D[(1,7)]=3`
- Iteration:

lists:

```
for i in A:
  print i
```

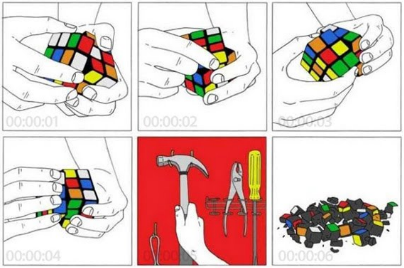
dictionaries:

```
for i in D:
  print i, D[i]
```

cycles:

```
i = 0
while i < 10:
  print i
  i += 1
```

Putting things together

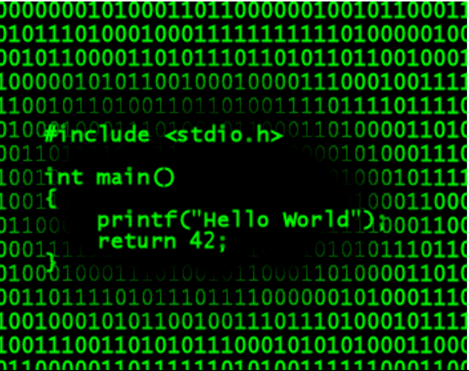


- import the `gurobipy` module
- create a `model` object
  - add variables
  - add constraints
- *[debug?]*
- solve
- report solution

U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Hello world example

$$
\begin{array}{lrcl}
\text{minimize} & 3000x + 4000y & & \\
\text{subject to:} & 5x + 6y & \geq & 10 \\
 & 7x + 5y & \geq & 5 \\
 & x, y & \geq & 0
\end{array}
$$

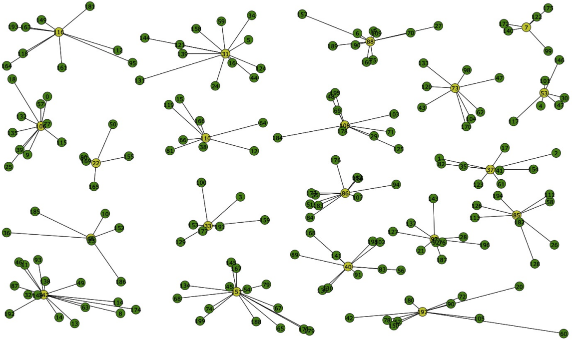



```
from gurobipy import *
model = Model("hello")

x = model.addVar(obj=3000, vtype="C", name="x")
y = model.addVar(obj=4000, vtype="C", name="y")
model.update()

L1 = LinExpr([5,6],[x,y])
model.addConstr(L1,">",10)
L2 = LinExpr([7,5],[x,y])
model.addConstr(L2,">",5)

model.ModelSense = 1                    # minimize
model.optimize()

if model.Status == GRB.OPTIMAL:
    print "Opt. Value=",model.ObjVal
    print "x* =", x.X
    print "y* =", y.X
```

U.PORTO
FC FACULDADE DE CIÊNCIAS UNIVERSIDADE DO PORTO

The k-median problem

- facility location problem of min-sum type
- n customers
- m positions for facilities (at some customer's coordinates)
- k maximum open facilities
- minimize service time summed for all the customers
- (Euclidean distance, random uniform (x, y) coordinates)



U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

The k-median problem — formulation

- n customers, m facilities
- variables:
  - $x_{ij} = 1$ if customer i is served by facility j
  - $y_j = 1$ if facility j is open

1. all customers must be served
2. maximum of k open facilities
3. customer i can be served by j only if j is open
4. minimize total, accumulated service time

$$
\begin{array}{lll}
\text{minimize} & \sum_i \sum_j c_{ij} x_{ij} & \\
\text{subject to} & \sum_j x_{ij} = 1 & \forall i \\
 & \sum_j y_j = k & \\
 & x_{ij} \leq y_j & \forall i, j \\
 & x_{ij} \in \{0, 1\} & \forall i \\
 & y_j \in \{0, 1\} & \forall j
\end{array}
$$

The *k*-median problem — Python/Gurobi model

```
def kmedian(m, n, c, k):
    model = Model("k-median")
    y,x = {}, {}
    for j in range(m):
        y[j] = model.addVar(obj=0, vtype="B", name="y[%s]"%j)
        for i in range(n):
            x[i,j] = model.addVar(obj=c[i,j], vtype="B", name="x[%s,%s]"%(i,j))
    model.update()

    for i in range(n):
        coef = [1 for j in range(m)]
        var = [x[i,j] for j in range(m)]
        model.addConstr(LinExpr(coef,var), "=", 1, name="Assign[%s]"%i)
    for j in range(m):
        for i in range(n):
            model.addConstr(x[i,j], "<", y[j], name="Strong[%s,%s]"%(i,j))
    coef = [1 for j in range(m)]
    var = [y[j] for j in range(m)]
    model.addConstr(LinExpr(coef,var), "=", rhs=k, name="k_median")

    model.update()
    model.__data = x,y
    return model
```

The *k*-median problem — preparing data

```
import math
import random
def distance(x1, y1, x2, y2):
    return math.sqrt((x2-x1)**2 + (y2-y1)**2)

def make_data(n):
    x = [random.random() for i in range(n)]
    y = [random.random() for i in range(n)]
    c = {}
    for i in range(n):
        for j in range(n):
            c[i,j] = distance(x[i],y[i],x[j],y[j])
    return c, x, y
```

The *k*-median problem — calling and solving

```
n = 200
c, x_pos, y_pos = make_data(n)
m = n
k = 20
model = kmedian(m, n, c, k)

model.optimize()
x,y = model.__data
edges = [(i,j) for (i,j) in x if x[i,j].X == 1]
nodes = [j for j in y if y[j].X == 1]
print "Optimal value=", model.ObjVal
print "Selected nodes:", nodes
print "Edges:", edges
```

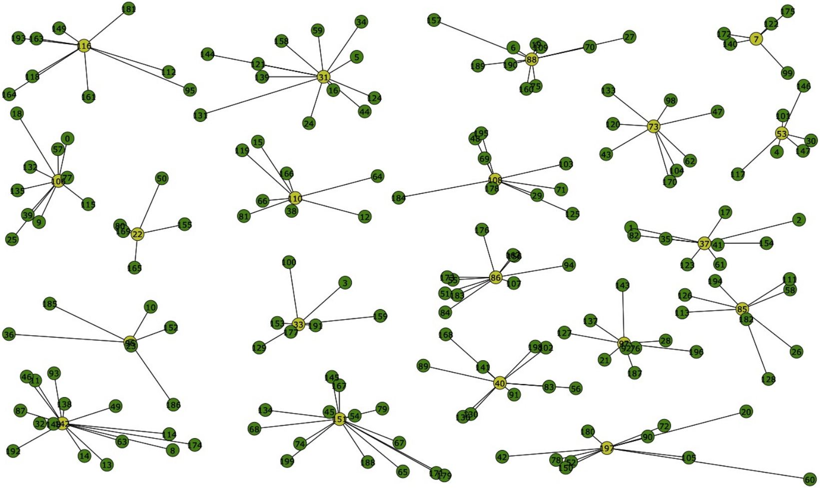
The k-median problem — plotting

```
import networkx as NX
import matplotlib.pyplot as P
P.ion() # interactive mode on
G = NX.Graph()

other = [j for j in y if j not in nodes]
G.add_nodes_from(nodes)
G.add_nodes_from(other)
for (i,j) in edges:
    G.add_edge(i,j)

position = {}
for i in range(n):
    position[i]=(x_pos[i],y_pos[i])

NX.draw(G, position, node_color='y', nodelist=nodes)
NX.draw(G, position, node_color='g', nodelist=other)
```

The *k*-median problem — solver output

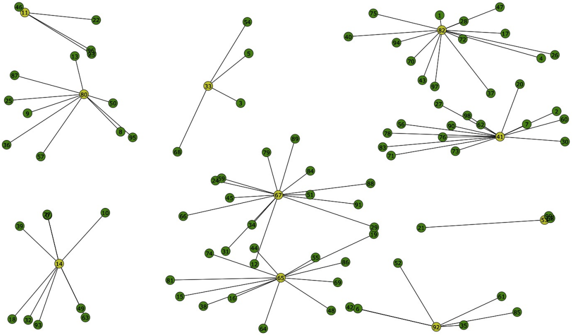
```
Optimize a model with 40201 rows, 40200 columns and 120200 nonzeros
Presolve time: 1.67s
Presolved: 40201 rows, 40200 columns, 120200 nonzeros
Variable types: 0 continuous, 40200 integer (40200 binary)
Found heuristic solution: objective 22.1688378
Root relaxation: objective 1.445152e+01, 2771 iterations, 0.55 seconds
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
     0     0   14.45152    0   92   22.16884   14.45152  34.8%     -    2s
H    0     0                      14.4528610   14.45152  0.01%     -    2s
Cutting planes:
  Gomory: 1
  Zero half: 1
Explored 0 nodes (2771 simplex iterations) in 2.67 seconds
Thread count was 1 (of 8 available processors)

Optimal solution found (tolerance 1.00e-04)
Best objective 1.445286097717e+01, best bound 1.445151681275e+01, gap 0.0093%
Optimal value= 14.4528609772
Selected nodes: [7, 22, 31, 33, 37, 40, 53, 73, 85, 86, 88, 96, 97, 106, 108, 110, 116, 142, 151, 197]
Edges: [(57, 106), (85, 85), (67, 151), (174, 142), (139, 31), (136, 40), (35, 37), (105, [illegible]), (195, 108),
max c: 0.257672494705
```

The k-median problem: solution



The *k*-center problem

- facility location problem of min-max type
- *n* customers
- *m* positions for facilities (at some customer's coordinates)
- *k* maximum open facilities
- minimize service time for the **latest-served** customer
- (Euclidean distance, random uniform (x, y) coordinates)



The k-center problem — formulation (min-max type)

- $x_{ij} = 1$ if customer i is served by facility j
- $y_j = 1$ if a facility j is open

1. all customers must be served
2. maximum of k open facilities
3. customer i can be served by j only if j is open
4. update service time for the latest-served customer

$$
\begin{array}{lll}
\text{minimize} & z & \\
\text{subject to} & \sum_j x_{ij} = 1 & \forall i \\
& \sum_j y_j = k & \\
& x_{ij} \leq y_j & \forall i, j \\
& c_{ij} x_{ij} \leq z & \forall i, j \\
& x_{ij} \in \{0, 1\} & \forall i, j \\
& y_j \in \{0, 1\} & \forall j
\end{array}
$$

The *k*-center problem — Python/Gurobi model

```
def kcenter(m, n, c, k):
    model = Model("k-center")
    z = model.addVar(obj=1, vtype="C", name="z")
    y, x = {}, {}
    for j in range(m):
        y[j] = model.addVar(obj=0, vtype="B", name="y[%s]"%j)
        for i in range(n):
            x[i,j] = model.addVar(obj=0, vtype="B", name="x[%s,%s]"%(i,j))
    model.update()
    for i in range(n):
        coef = [1 for j in range(m)]
        var = [x[i,j] for j in range(m)]
        model.addConstr(LinExpr(coef,var), "=", 1, name="Assign[%s]"%i)
    for j in range(m):
        for i in range(n):
            model.addConstr(x[i,j], "<", y[j], name="Strong[%s,%s]"%(i,j))
    for i in range(n):
        for j in range(n):
            model.addConstr(LinExpr(c[i,j],x[i,j]), "<", z, name="Max_x[%s,%s]"%(i,j))
    coef = [1 for j in range(m)]
    var = [y[j] for j in range(m)]
    model.addConstr(LinExpr(coef,var), "=", rhs=k, name="k_center")
    model.update()
    model.__data = x,y
    return model
```

The k-center problem — solver output

```
Optimize a model with 20101 rows, 10101 columns and 50000 nonzeros
Presolve removed 100 rows and 0 columns
Presolve time: 0.35s
Presolved: 20001 rows, 10101 columns, 49900 nonzeros
Variable types: 1 continuous, 10100 integer (10100 binary)
Found heuristic solution: objective 0.9392708
Found heuristic solution: objective 0.9388764
Found heuristic solution: objective 0.9335182
Root relaxation: objective 3.637572e-03, 13156 iterations, 1.88 seconds
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
     0     0    0.00364    0 9255    0.93352    0.00364  100%     -    3s
[...]
H    7     0                    0.2187034    0.21870  0.0%   603  454s
Cutting planes:
  Gomory: 1
  Zero half: 2
Explored 7 nodes (83542 simplex iterations) in 454.11 seconds
Thread count was 1 (of 8 available processors)

Optimal solution found (tolerance 1.00e-04)
Best objective 2.187034280810e-01, best bound 2.187034280810e-01, gap 0.0%
Optimal value= 0.218703428081
Selected nodes: [12, 14, 23, 33, 41, 51, 53, 72, 80, 92]
Edges: [(53, 53), (36, 80), (54, 33), (69, 12), (39, 14), (86, 51), (99, 53), (37, 41), (49, 14), (26, 72), (2
```

CPU usage

- k-median instance: $n = m = 200$, $k = 20$, CPU = 5s
- k-center instance: $n = m = 100$, $k = 10$, CPU = 454s
- k-center: for an instance that is **half size** the one solved for k-median, used **almost ten times** more CPU
- can we do better?

The k-center problem — formulation (min type)

- $a_{ij} = 1$ if customer i **can be** served by facility j
- $y_j = 1$ if a facility j is open
- $\xi_i = 1$ if customer i cannot be served
- parameter: distance θ for which a client can be served
    - if $c_{ij} < \theta$ then set $a_{ij} = 1$
    - else, set $a_{ij} = 1$

1. either customer i is served or $\xi = 1$
2. maximum of k open facilities

$$
\begin{array}{lll}
\text{minimize} & \sum_i \xi_i & \\
\text{subject to} & \sum_j a_{ij} y_j + \xi_i \geq 1 & \forall i \\
& \sum_j y_j = k & \\
& \xi_i \in \{0, 1\} & \forall i \\
& y_j \in \{0, 1\} & \forall j
\end{array}
$$

U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

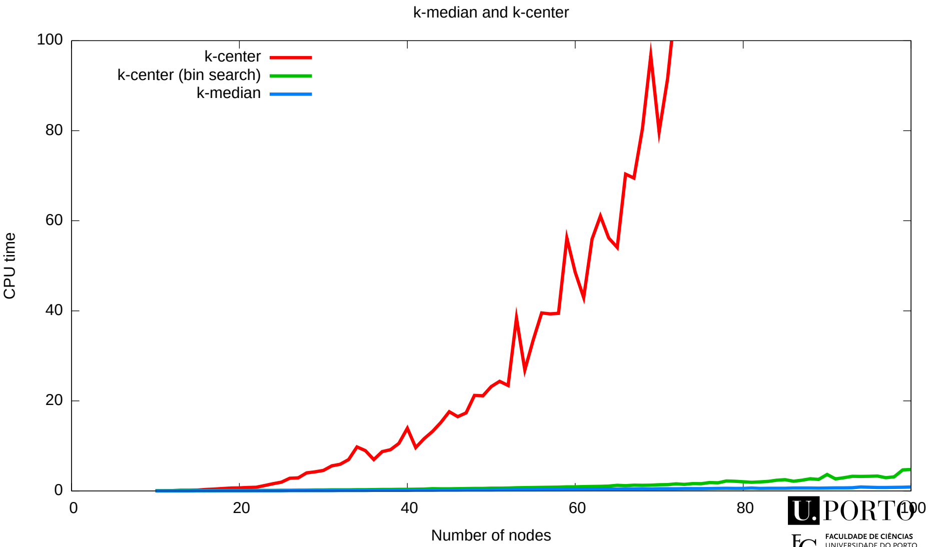
The *k*-center problem — model for binary search

```
def kcenter(m, n, c, k, max_c):
    model = Model("k-center")
    z, y, x = {}, {}, {}
    for i in range(n):
        z[i] = model.addVar(obj=1, vtype="B", name="z[%s]"%i)
    for j in range(m):
        y[j] = model.addVar(obj=0, vtype="B", name="y[%s]"%j)
        for i in range(n):
            x[i,j] = model.addVar(obj=0, vtype="B", name="x[%s,%s]"%(i,j))
    model.update()
    for i in range(n):
        coef = [1 for j in range(m)]
        var = [x[i,j] for j in range(m)]
        var.append(z[i])
        model.addConstr(LinExpr(coef,var), "=", 1, name="Assign[%s]"%i)
    for j in range(m):
        for i in range(n):
            model.addConstr(x[i,j], "<", y[j], name="Strong[%s,%s]"%(i,j))
    coef = [1 for j in range(m)]
    var = [y[j] for j in range(m)]
    model.addConstr(LinExpr(coef,var), "=", rhs=k, name="k_center")
    model.update()
    model.__data = x,y,z
    return model
```

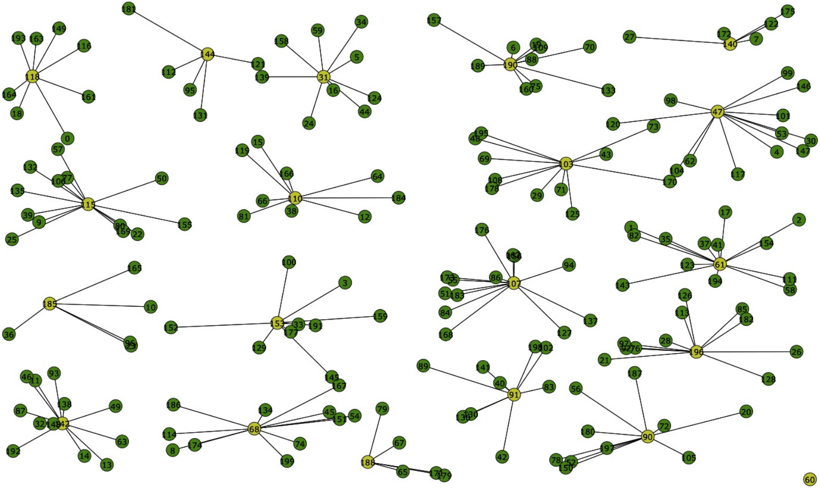
The *k*-center problem — binary search

```
def solve_kcenter(m, n, c, k, max_c, delta):
    model = kcenter(m, n, c, k, max_c)
    x,y,z = model.__data
    LB = 0
    UB = max_c
    while UB-LB > delta:
        theta = (UB+LB) / 2.
        for j in range(m):
            for i in range(n):
                if c[i,j]>theta:
                    x[i,j].UB = 0
                else:
                    x[i,j].UB = 1.0
        model.update()
        model.optimize()
        infeasibility = sum([z[i].X for i in range(m)])
        if infeasibility > 0:
            LB = theta
        else:
            UB = theta
            nodes = [j for j in y if y[j].X == 1]
            edges = [(i,j) for (i,j) in x if x[i,j].X == 1]
    return nodes, edges
```

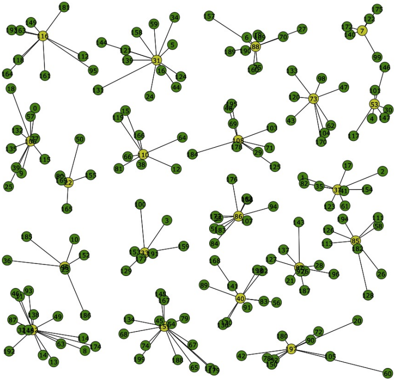
The k-center problem: CPU usage

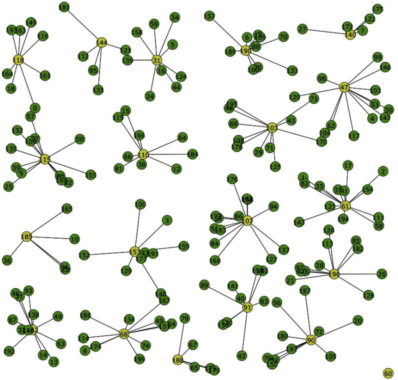


The *k*-center problem: solution



The k-median (left) and k-center (right) solutions





U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO