Masterclass

May 13, 2021

Combinatorial Optimization course, FEE CTU in Prague. Created by Industrial Informatics Department.

[1]:
```
%pip install -i https://pypi.gurobi.com gurobipy
import gurobipy as g
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
```

```
Looking in indexes: https://pypi.gurobi.com
Requirement already satisfied: gurobipy in
/home/benedond/anaconda3/lib/python3.7/site-packages (9.1.0)
Note: you may need to restart the kernel to use updated packages.
```

1 Automatized model tuning

As an example, we will revisit the model for Game of Fiver which for larger board sizes took considerable time to solve:

[2]:
```
def game_of_fivers(n, params=None):
  m = g.Model()

  x = m.addVars(n+2, n+2, vtype=g.GRB.BINARY, obj=1)
  k = m.addVars(range(1,n+1), range(1,n+1), vtype=g.GRB.INTEGER)

  for i in range(1,n+1):
      for j in range(1,n+1):
          m.addConstr(x[i,j] + x[i+1, j] + x[i-1,j] + x[i,j+1] + x[i,j-1] ==␣
↪2*k[i,j] + 1)

  m.addConstr(x.sum(0,"*") + x.sum(n+1,"*") + x.sum("*",0) + x.sum("*",n+1) ==␣
↪0)

  m.write('fivers_n{}.lp'.format(n))

  if params is not None:
    m.params.CutPasses = 10
```

```
    m.params.PreDual = 1
    m.params.outputflag = 0

  m.optimize()

  X = [[int(round(x[i,j].X)) for j in range(1,n+1)] for i in range(1,n+1)]
  return m.runtime
```

[3]:
```
# run with default parameters
def_time = game_of_fivers(21)
print('Time with default settings: {}s'.format(def_time))
```

```
Using license file /home/benedond/Apps/gurobi/gurobi910/gurobi.lic
Academic license - for non-commercial use only - expires 2021-06-18
Gurobi Optimizer version 9.1.0 build v9.1.0rc0 (linux64)
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads
Optimize a model with 442 rows, 970 columns and 2734 nonzeros
Model fingerprint: 0xaa07108c
Variable types: 0 continuous, 970 integer (529 binary)
Coefficient statistics:
  Matrix range     [1e+00, 2e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 1 rows and 96 columns
Presolve time: 0.01s
Presolved: 441 rows, 874 columns, 2533 nonzeros
Variable types: 0 continuous, 874 integer (518 binary)

Root relaxation: objective 9.585736e+01, 1221 iterations, 0.06 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   95.85736    0  440          -   95.85736      -     -    0s
     0     0   96.94696    0  468          -   96.94696      -     -    0s
     0     0   96.98572    0  464          -   96.98572      -     -    0s
     0     0   96.98726    0  465          -   96.98726      -     -    0s
     0     0   96.98919    0  470          -   96.98919      -     -    0s
     0     0   96.98928    0  471          -   96.98928      -     -    0s
     0     0   97.67888    0  479          -   97.67888      -     -    0s
     0     0   97.71186    0  478          -   97.71186      -     -    0s
     0     0   97.71283    0  479          -   97.71283      -     -    0s
     0     0   98.44142    0  485          -   98.44142      -     -    0s
     0     0   98.51498    0  491          -   98.51498      -     -    0s
     0     0   98.51785    0  492          -   98.51785      -     -    0s
     0     0   99.37022    0  490          -   99.37022      -     -    0s
```

```
     0     0   99.45836    0  495          -   99.45836      -     -    0s
     0     0   99.48538    0  498          -   99.48538      -     -    0s
     0     0   99.49044    0  500          -   99.49044      -     -    0s
     0     0   99.49082    0  503          -   99.49082      -     -    0s
     0     0  100.38463    0  506          -  100.38463      -     -    0s
     0     0  100.69590    0  511          -  100.69590      -     -    0s
     0     0  101.33193    0  520          -  101.33193      -     -    0s
     0     0  102.31607    0  529          -  102.31607      -     -    0s
     0     0  102.60194    0  538          -  102.60194      -     -    0s
     0     0  103.02475    0  529          -  103.02475      -     -    0s
     0     0  103.51409    0  536          -  103.51409      -     -    1s
     0     0  103.65124    0  543          -  103.65124      -     -    1s
     0     0  103.98002    0  541          -  103.98002      -     -    1s
     0     0  104.68522    0  551          -  104.68522      -     -    1s
     0     0  105.01278    0  553          -  105.01278      -     -    1s
     0     0  105.20101    0  565          -  105.20101      -     -    1s
     0     0  105.20101    0  565          -  105.20101      -     -    1s
     0     2  105.21072    0  565          -  105.21072      -     -    1s
   493   413  108.80917   11  525          -  105.44028      -   142    5s
   793   622  114.07780   26  655          -  114.07780      -   131   10s
   833   649  117.40460   30  663          -  117.40460      -   125   15s
   906   698  118.68468   46  747          -  118.68468      -   115   20s
   982   748  122.43603   31  749          -  119.20375      -   106   25s
H 1007   723                     245.0000000  119.31481  51.3%   131   26s

Cutting planes:
  MIR: 259
  Flow cover: 50
  Zero half: 132

Explored 1007 nodes (137125 simplex iterations) in 26.90 seconds
Thread count was 4 (of 4 available processors)

Solution count 1: 245

Optimal solution found (tolerance 1.00e-04)
Best objective 2.450000000000e+02, best bound 2.450000000000e+02, gap 0.0000%
Time with default settings: 26.91558289527893s
```

[4]:
```python
# lets try to tune some of the params
model = g.read('fivers_n21.lp')

model.params.tuneResults = 1
model.params.TuneTimeLimit = 30   # how much time to invest into the tuning

model.tune()
if model.tuneResultCount > 0:
```

```
    model.getTuneResult(0)
    model.write('fivers_tuned_params.prm')
```

```
Read LP format model from file fivers_n21.lp
Reading time = 0.00 seconds
: 442 rows, 970 columns, 2734 nonzeros
Changed value of parameter tuneResults to 1
   Prev: -1  Min: -1  Max: 2000000000  Default: -1
Changed value of parameter TuneTimeLimit to 30.0
   Prev: -1.0  Min: -1.0  Max: inf  Default: -1.0

Solving model using baseline parameter set with TimeLimit=3s

Testing candidate parameter set 1…

        Default parameters

Solving with random seed #1 …
Optimize a model with 442 rows, 970 columns and 2734 nonzeros
Model fingerprint: 0xd83b2255
Variable types: 0 continuous, 970 integer (529 binary)
Coefficient statistics:
  Matrix range     [1e+00, 2e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 1 rows and 96 columns
Presolve time: 0.01s
Presolved: 441 rows, 874 columns, 2533 nonzeros
Variable types: 0 continuous, 874 integer (518 binary)

Root relaxation: objective 9.585736e+01, 1221 iterations, 0.06 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   95.85736    0  440          -   95.85736      -     -    0s
     0     0   96.94696    0  468          -   96.94696      -     -    0s
     0     0   96.98572    0  464          -   96.98572      -     -    0s
     0     0   96.98726    0  465          -   96.98726      -     -    0s
     0     0   96.98919    0  470          -   96.98919      -     -    0s
     0     0   96.98928    0  471          -   96.98928      -     -    0s
     0     0   97.67888    0  479          -   97.67888      -     -    0s
     0     0   97.71186    0  478          -   97.71186      -     -    0s
     0     0   97.71283    0  479          -   97.71283      -     -    0s
     0     0   98.44142    0  485          -   98.44142      -     -    0s
     0     0   98.51498    0  491          -   98.51498      -     -    0s
```

```
     0     0   98.51785    0  492          -   98.51785      -     -    0s
     0     0   99.37022    0  490          -   99.37022      -     -    0s
     0     0   99.45836    0  495          -   99.45836      -     -    0s
     0     0   99.48538    0  498          -   99.48538      -     -    0s
     0     0   99.49044    0  500          -   99.49044      -     -    0s
     0     0   99.49082    0  503          -   99.49082      -     -    0s
     0     0  100.38463    0  506          -  100.38463      -     -    0s
     0     0  100.69590    0  511          -  100.69590      -     -    0s
     0     0  101.33193    0  520          -  101.33193      -     -    0s
     0     0  102.31607    0  529          -  102.31607      -     -    0s
     0     0  102.60194    0  538          -  102.60194      -     -    0s
     0     0  103.02475    0  529          -  103.02475      -     -    0s
     0     0  103.51409    0  536          -  103.51409      -     -    1s
     0     0  103.65124    0  543          -  103.65124      -     -    1s
     0     0  103.98002    0  541          -  103.98002      -     -    1s
     0     0  104.68522    0  551          -  104.68522      -     -    1s
     0     0  105.01278    0  553          -  105.01278      -     -    1s
     0     0  105.20101    0  565          -  105.20101      -     -    1s
     0     0  105.20101    0  565          -  105.20101      -     -    1s
     0     2  105.21072    0  565          -  105.21072      -     -    1s

Cutting planes:
  Cover: 2
  Implied bound: 1
  MIR: 110
  Zero half: 67
  RLT: 2

Explored 197 nodes (32326 simplex iterations) in 3.01 seconds
Thread count was 4 (of 4 available processors)

Solution count 0

Time limit reached
Best objective -, best bound 1.060000000000e+02, gap -

------------------------------------------------------------------------------

Solving with random seed #2 …
Optimize a model with 442 rows, 970 columns and 2734 nonzeros
Model fingerprint: 0xd83b2255
Variable types: 0 continuous, 970 integer (529 binary)
Coefficient statistics:
  Matrix range     [1e+00, 2e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 1 rows and 96 columns
```

```
Presolve time: 0.01s
Presolved: 441 rows, 874 columns, 2533 nonzeros
Variable types: 0 continuous, 874 integer (518 binary)

Root relaxation: objective 9.585736e+01, 1219 iterations, 0.06 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   95.85736    0  440          -   95.85736      -     -    0s
     0     0   96.93040    0  461          -   96.93040      -     -    0s
     0     0   97.04389    0  467          -   97.04389      -     -    0s
     0     0   97.05700    0  466          -   97.05700      -     -    0s
     0     0   97.05720    0  467          -   97.05720      -     -    0s
     0     0   98.36331    0  475          -   98.36331      -     -    0s
     0     0   98.42652    0  478          -   98.42652      -     -    0s
     0     0   98.44018    0  481          -   98.44018      -     -    0s
     0     0   98.44254    0  482          -   98.44254      -     -    0s
     0     0   98.44382    0  483          -   98.44382      -     -    0s
     0     0   98.44385    0  484          -   98.44385      -     -    0s
     0     0   99.19140    0  489          -   99.19140      -     -    0s
     0     0   99.25976    0  489          -   99.25976      -     -    0s
     0     0   99.29452    0  492          -   99.29452      -     -    0s
     0     0   99.29710    0  494          -   99.29710      -     -    0s
     0     0   99.29844    0  497          -   99.29844      -     -    0s
     0     0   99.61365    0  499          -   99.61365      -     -    0s
     0     0   99.74041    0  500          -   99.74041      -     -    0s
     0     0   99.75620    0  506          -   99.75620      -     -    0s
     0     0   99.75901    0  506          -   99.75901      -     -    0s
     0     0  100.37690    0  502          -  100.37690      -     -    0s
     0     0  100.51491    0  509          -  100.51491      -     -    0s
     0     0  100.52406    0  510          -  100.52406      -     -    0s
     0     0  100.52712    0  513          -  100.52712      -     -    0s
     0     0  100.83827    0  515          -  100.83827      -     -    0s
     0     0  100.83909    0  518          -  100.83909      -     -    0s
     0     0  101.90654    0  509          -  101.90654      -     -    0s
     0     0  102.66904    0  519          -  102.66904      -     -    0s
     0     0  103.01378    0  527          -  103.01378      -     -    0s
     0     0  103.65683    0  532          -  103.65683      -     -    1s
     0     0  104.33427    0  536          -  104.33427      -     -    1s
     0     0  105.34836    0  534          -  105.34836      -     -    1s
     0     0  105.93595    0  558          -  105.93595      -     -    1s
     0     0  106.55636    0  561          -  106.55636      -     -    1s
     0     0  106.85965    0  567          -  106.85965      -     -    1s
     0     0  107.17770    0  566          -  107.17770      -     -    1s
     0     0  107.51395    0  567          -  107.51395      -     -    1s
     0     0  107.94968    0  580          -  107.94968      -     -    1s
     0     0  108.36590    0  578          -  108.36590      -     -    1s
```

```
     0     0  108.88869    0  585          -  108.88869      -     -    1s
     0     0  109.36345    0  590          -  109.36345      -     -    1s
     0     0  109.81196    0  587          -  109.81196      -     -    2s
     0     0  110.10839    0  593          -  110.10839      -     -    2s
     0     0  110.10839    0  592          -  110.10839      -     -    2s
     0     2  110.11907    0  592          -  110.11907      -     -    2s

Cutting planes:
  Cover: 2
  Implied bound: 2
  MIR: 93
  Zero half: 101
  RLT: 1

Explored 76 nodes (18148 simplex iterations) in 3.01 seconds
Thread count was 4 (of 4 available processors)

Solution count 0

Time limit reached
Best objective -, best bound 1.110000000000e+02, gap -

--------------------------------------------------------------------------------

Solving with random seed #3 …
Optimize a model with 442 rows, 970 columns and 2734 nonzeros
Model fingerprint: 0xd83b2255
Variable types: 0 continuous, 970 integer (529 binary)
Coefficient statistics:
  Matrix range     [1e+00, 2e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 1 rows and 96 columns
Presolve time: 0.01s
Presolved: 441 rows, 874 columns, 2533 nonzeros
Variable types: 0 continuous, 874 integer (518 binary)

Root relaxation: objective 9.585736e+01, 1228 iterations, 0.05 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   95.85736    0  440          -   95.85736      -     -    0s
     0     0   96.97025    0  463          -   96.97025      -     -    0s
     0     0   97.08247    0  464          -   97.08247      -     -    0s
     0     0   97.09562    0  467          -   97.09562      -     -    0s
     0     0   97.09796    0  467          -   97.09796      -     -    0s
```

```
     0     0   97.09853    0  466          -   97.09853      -     -    0s
     0     0   97.09853    0  467          -   97.09853      -     -    0s
     0     0   98.07699    0  472          -   98.07699      -     -    0s
     0     0   98.19686    0  480          -   98.19686      -     -    0s
     0     0   98.20215    0  482          -   98.20215      -     -    0s
     0     0   98.20253    0  482          -   98.20253      -     -    0s
     0     0   98.66571    0  489          -   98.66571      -     -    0s
     0     0   98.72609    0  491          -   98.72609      -     -    0s
     0     0   98.73460    0  495          -   98.73460      -     -    0s
     0     0   98.74101    0  499          -   98.74101      -     -    0s
     0     0   98.74154    0  501          -   98.74154      -     -    0s
     0     0  100.07621    0  497          -  100.07621      -     -    0s
     0     0  100.09689    0  498          -  100.09689      -     -    0s
     0     0  100.09785    0  497          -  100.09785      -     -    0s
     0     0  100.88550    0  512          -  100.88550      -     -    0s
     0     0  101.78031    0  512          -  101.78031      -     -    0s
     0     0  102.31611    0  520          -  102.31611      -     -    0s
     0     0  103.06033    0  531          -  103.06033      -     -    0s
     0     0  104.07059    0  523          -  104.07059      -     -    0s
     0     0  104.67974    0  549          -  104.67974      -     -    0s
     0     0  105.52274    0  550          -  105.52274      -     -    1s
     0     0  106.03969    0  562          -  106.03969      -     -    1s
     0     0  106.38329    0  572          -  106.38329      -     -    1s
     0     0  106.58346    0  578          -  106.58346      -     -    1s
     0     0  106.58346    0  577          -  106.58346      -     -    1s
     0     2  106.59237    0  575          -  106.59237      -     -    1s

Cutting planes:
  Cover: 2
  MIR: 105
  StrongCG: 2
  Zero half: 79
  RLT: 1

Explored 221 nodes (33883 simplex iterations) in 3.01 seconds
Thread count was 4 (of 4 available processors)

Solution count 0

Time limit reached
Best objective -, best bound 1.070000000000e+02, gap -


------------------------------------------------------------------------------
Begin tuning (baseline 3 no_solution)…
------------------------------------------------------------------------------

Testing candidate parameter set 2…
```

```
        Heuristics 0.5

Solving with random seed #1 … MIP gap 56.7%
Solving with random seed #2 … MIP gap 54.7%
Solving with random seed #3 … MIP gap 56.3%

Improvement found:
  baseline: 3 no_solution
  improved: mean MIP gap 55.9%
Total elapsed tuning time 18s (12s remaining)

---------------------------------------------------------------------------

Testing candidate parameter set 3…

        MIPFocus 2

Solving with random seed #1 … MIP gap -

Progress so far:
  baseline: 3 no_solution
  best:     mean MIP gap 55.9%
Total elapsed tuning time 21s (9s remaining)

---------------------------------------------------------------------------

Testing candidate parameter set 4…

        Heuristics 0.5
        ZeroHalfCuts 1

Solving with random seed #1 … MIP gap 56.7%
Solving with random seed #2 … MIP gap 54.7%
Solving with random seed #3 …

Tune time limit reached.

---------------------------------------------------------------------------

Tested 3 parameter sets in 30.00s

Baseline parameter set: 3 no_solution

        Default parameters

 # Name              0        1        2      Avg  Std Dev
 0 Model             -        -        -        -        -
```

```
Improved parameter set 1 (mean MIP gap 55.9%):

        Heuristics 0.5

 # Name              0        1        2      Avg  Std Dev
 0 Model         56.7%    54.7%    56.3%    55.9%     0.88
```

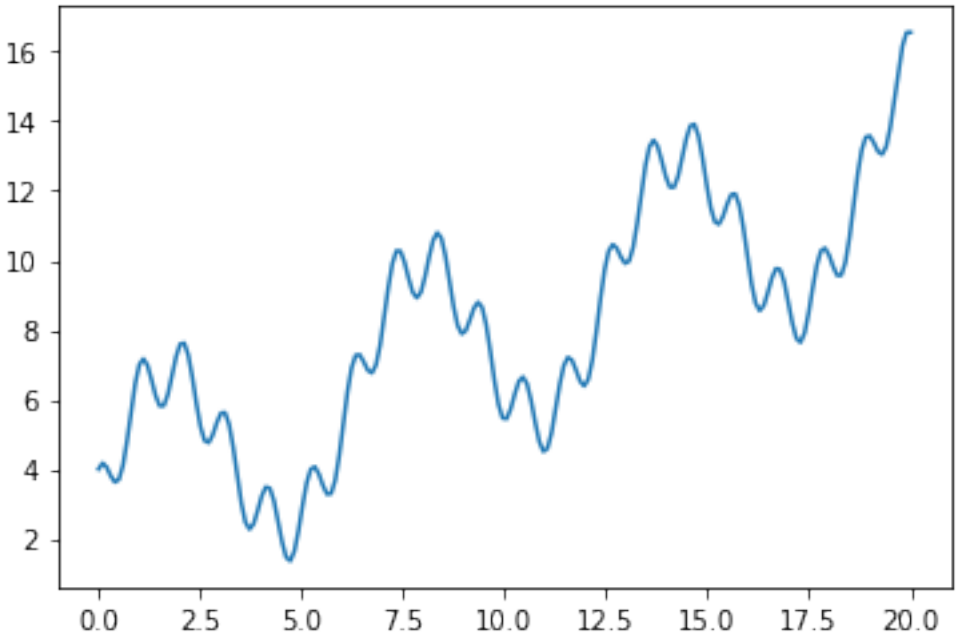
[5]:
```
print('Time before tuning: {}s'.format(def_time))
print('Time after tuning: {}s'.format(game_of_fivers(21, {'CutPasses': 10,␣
 ↪'PreDual':1})))
```

```
Time before tuning: 26.91558289527893s
Changed value of parameter CutPasses to 10
   Prev: -1  Min: -1  Max: 2000000000  Default: -1
Changed value of parameter PreDual to 1
   Prev: -1  Min: -1  Max: 2  Default: -1
Time after tuning: 8.29596495628357s
```

2 Nonlinear constrains

[6]:
```
n = 200
t = np.linspace(0, 20, n)
y = 3*np.sin(t)+np.cos(6*t)+0.5*t+3

plt.plot(t, y)
```

[6]:
```
[<matplotlib.lines.Line2D at 0x7f2d77bb2f90>]
```

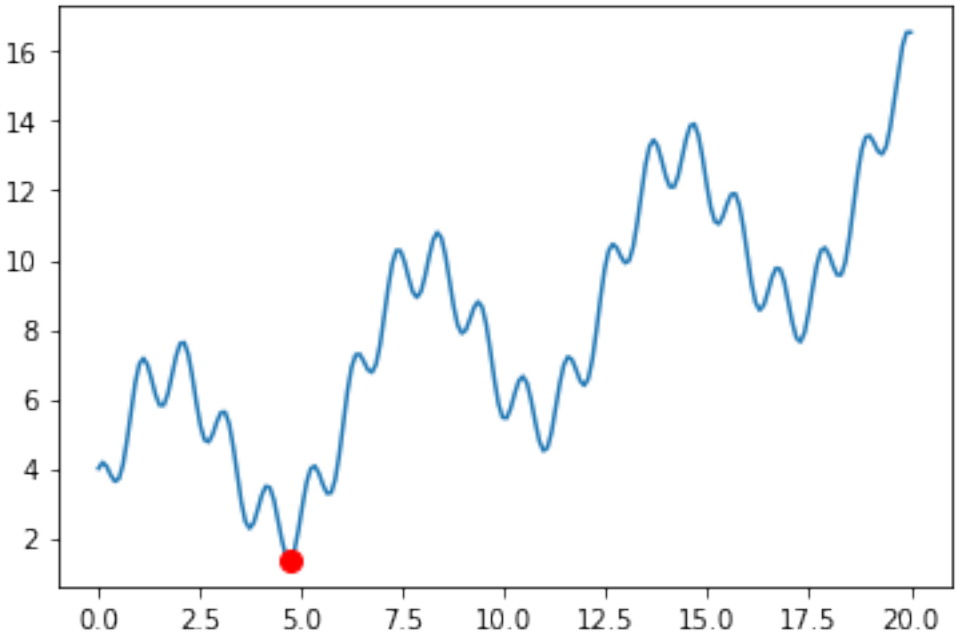


[7]:
```
m = g.Model()

u = m.addVar(vtype=g.GRB.CONTINUOUS)
v = m.addVar(vtype=g.GRB.CONTINUOUS)
m.addGenConstrPWL(u, v, t, y)

m.setObjective(v)

m.optimize()


plt.plot(t, y)
plt.plot(u.x, v.x, marker='o', markersize=8, color="red")
```

```
Gurobi Optimizer version 9.1.0 build v9.1.0rc0 (linux64)
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads
Optimize a model with 0 rows, 2 columns and 0 nonzeros
Model fingerprint: 0x8d885f17
Model has 1 general constraint
Variable types: 2 continuous, 0 integer (0 binary)
Coefficient statistics:
  Matrix range     [0e+00, 0e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [0e+00, 0e+00]
Presolve added 1 rows and 197 columns
```

```
Presolve time: 0.00s
Presolved: 1 rows, 199 columns, 199 nonzeros
Presolved model has 1 SOS constraint(s)
Variable types: 199 continuous, 0 integer (0 binary)

Root relaxation: objective 1.364267e+00, 0 iterations, 0.00 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

*    0     0               0       1.3642670    1.36427  0.00%     -    0s

Explored 0 nodes (0 simplex iterations) in 0.01 seconds
Thread count was 4 (of 4 available processors)

Solution count 1: 1.36427

Optimal solution found (tolerance 1.00e-04)
Best objective 1.364266996602e+00, best bound 1.364266996602e+00, gap 0.0000%
```

[7]: `[<matplotlib.lines.Line2D at 0x7f2d7609a610>]`

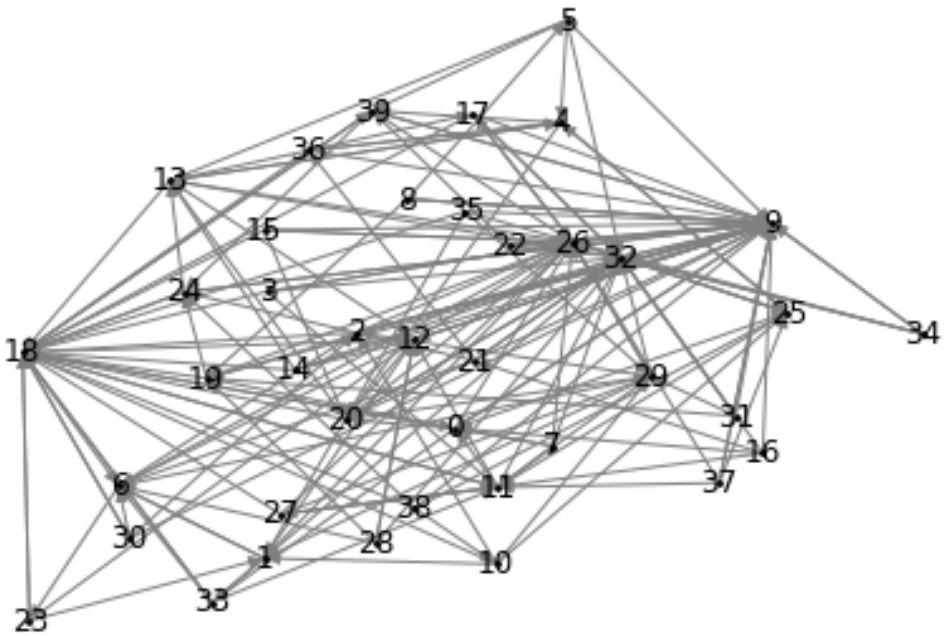


[ ]:

3 Solution pool

[8]:
```
dg = nx.DiGraph()
dg = nx.random_k_out_graph(40, 5, 0.4, seed=22)
pos = nx.spring_layout(dg)
nx.draw(dg, pos, node_color='k', node_size=3, edge_color='grey',␣
 ↪with_labels=True)
```

```
/home/benedond/anaconda3/lib/python3.7/site-
packages/networkx/drawing/nx_pylab.py:579: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in
3.3. Use np.iterable instead.
  if not cb.iterable(width):
/home/benedond/anaconda3/lib/python3.7/site-
packages/networkx/drawing/nx_pylab.py:676: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in
3.3. Use np.iterable instead.
  if cb.iterable(node_size):  # many node sizes
```



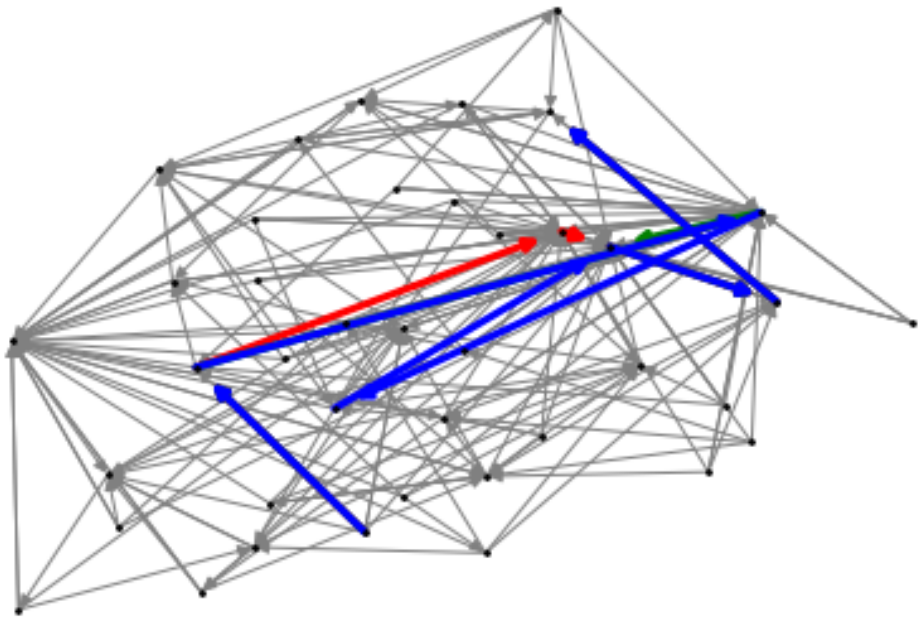
[9]:
```
s = 28
t = 4
E = dg.edges()
E = dict.fromkeys(E)
```

```
V = dg.nodes()
np.random.seed(69)
w = np.random.randint(0, 50, len(E))

m = g.Model()
x = m.addVars(E.keys(), vtype=g.GRB.BINARY, ub=1, obj=w)

m.addConstr(g.quicksum([x[s, j] for k, j in E if k == s]) == 1)
m.addConstr(g.quicksum([x[i, t] for i, k in E if k == t]) == 1)

for i in V:
  if i not in [s, t]:
    m.addConstr(g.quicksum([x[i, j] for k, j in E if k == i]) == g.
 ↪quicksum([x[j, i] for j, k in E if k == i]))

m.setParam(g.GRB.Param.PoolSolutions, 3)
m.setParam(g.GRB.Param.PoolSearchMode, 2)   # k-best solutions
m.optimize()

sols = [0]*m.solcount
colorlist = ['r', 'g', 'b']

print('Found {} solutions.'.format(m.solcount))
for sol_idx in range(m.solcount):
  print('Sol no. {}'.format(sol_idx+1))
  sols[sol_idx] = []
  m.setParam(g.GRB.Param.SolutionNumber, sol_idx)
  for i, j in E:
    if x[i, j].xn > 0.5:
      print(i, j)
      sols[sol_idx] += [(i, j)]

nx.draw(dg, pos, node_color='k', node_size=3, edge_color='grey')

for k in range(m.solcount):
  nx.draw_networkx_edges(dg, pos, edgelist=sols[k], edge_color=colorlist[k],␣
 ↪width=3)
```

```
Changed value of parameter PoolSolutions to 3
   Prev: 10  Min: 1  Max: 2000000000  Default: 10
Changed value of parameter PoolSearchMode to 2
   Prev: 0  Min: 0  Max: 2  Default: 0
Gurobi Optimizer version 9.1.0 build v9.1.0rc0 (linux64)
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads
Optimize a model with 40 rows, 174 columns and 331 nonzeros
Model fingerprint: 0x0bb97d12
Variable types: 0 continuous, 174 integer (174 binary)
```

```
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 5e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 314.0000000
Presolve removed 17 rows and 73 columns
Presolve time: 0.00s
Presolved: 23 rows, 101 columns, 185 nonzeros
Variable types: 0 continuous, 101 integer (101 binary)
Found heuristic solution: objective 43.0000000

Root relaxation: objective 4.100000e+01, 13 iterations, 0.00 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

*    0     0               0      41.0000000   41.00000  0.00%     -    0s

Optimal solution found at node 0 - now completing solution pool…

    Nodes    |    Current Node    |      Pool Obj. Bounds     |     Work
             |                    |   Worst                   |
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0          -    0        43.00000   41.00000  4.65%     -    0s
     0     0          -    0        43.00000   41.00000  4.65%     -    0s
     0     2          -    0        43.00000   41.00000  4.65%     -    0s

Explored 76 nodes (80 simplex iterations) in 0.05 seconds
Thread count was 4 (of 4 available processors)

Solution count 3: 41 42 42
No other solutions better than 42

Optimal solution found (tolerance 1.00e-04)
Best objective 4.100000000000e+01, best bound 4.100000000000e+01, gap 0.0000%
Found 3 solutions.
Sol no. 1
19 26
25 4
26 32
28 19
32 25
Sol no. 2
9 32
19 9
25 4
```

```
28 19
32 25
Sol no. 3
9 20
19 9
20 32
25 4
28 19
32 25
```



4 Bratley revisited: CP model

[10]:
```python
# Visualization
import matplotlib.pyplot as plt

def plot_solution(s, p):
    """
    s: solution vector
    p: processing times
    """
    fig = plt.figure(figsize=(10,2))
```

```
    ax = plt.gca()
    ax.set_xlabel('time')
    ax.grid(True)
    ax.set_yticks([2.5])
    ax.set_yticklabels(["oven"])
    eps = 0.25  # just to show spaces between the dishes
    ax.broken_barh([(s[i], p[i]-eps) for i in range(len(s))], (0, 5),
                   facecolors=('tab:orange', 'tab:green', 'tab:red', 'tab:
↪blue', 'tab:gray'))
```

[27]:
```
# Load data
path = "./bratley_data/instances/public_3.txt"

with open(path, "r") as f_in:
    lines = f_in.readlines()

    n = int(lines[0].strip())
    r,d,p = [],[],[]

    for i in range(n):
        (pi, ri, di) = list(map(int, lines[1+i].split()))
        r.append(ri)
        p.append(pi)
        d.append(di)

print("r", r, "d", d, "p", p, sep="\n")
```

```
r
[0, 4, 4, 4, 10, 9, 2, 1, 3, 1, 0, 4]
d
[10, 39, 39, 39, 30, 30, 12, 49, 7, 20, 39, 39]
p
[3, 2, 2, 2, 1, 4, 5, 5, 3, 5, 3, 2]
```

[11]:
```
# Generate data
from numpy import random as rnd
import numpy as np
rnd.seed(15)

n = 200
p = [rnd.randint(1,100) for i in range(n)]

r = [0 for i in range(n)]
for i in range(1,n):
    r[i] = int(round(r[i-1] + rnd.exponential(0.5* sum(p)/len(p))))
```

```
d = [ int(round(r[i] + p[i] + rnd.exponential(100*sum(p)/len(p)) )) for i in␣
 ↪range(n)]
print("r", r, "d", d, "p", p, sep="\n")
```

```
r
[0, 17, 47, 59, 64, 67, 68, 88, 114, 191, 191, 279, 287, 288, 294, 318, 350,
460, 476, 509, 573, 584, 589, 620, 641, 792, 793, 872, 929, 949, 972, 997, 1006,
1031, 1069, 1143, 1145, 1145, 1151, 1167, 1170, 1189, 1191, 1193, 1227, 1244,
1327, 1363, 1371, 1446, 1549, 1557, 1576, 1586, 1646, 1665, 1677, 1711, 1711,
1722, 1742, 1759, 1761, 1778, 1803, 1832, 1840, 1844, 1899, 1938, 1947, 1967,
2018, 2109, 2123, 2288, 2300, 2306, 2312, 2333, 2337, 2370, 2461, 2530, 2535,
2560, 2580, 2613, 2645, 2668, 2696, 2710, 2761, 2801, 2825, 2864, 2871, 2875,
2880, 2884, 2890, 2891, 2955, 2978, 2990, 3072, 3113, 3138, 3169, 3216, 3269,
3277, 3312, 3343, 3431, 3445, 3483, 3483, 3517, 3531, 3581, 3623, 3649, 3656,
3662, 3671, 3681, 3683, 3698, 3708, 3759, 3768, 3770, 3830, 3838, 3913, 3921,
3978, 3982, 4015, 4018, 4025, 4046, 4073, 4099, 4149, 4163, 4213, 4220, 4244,
4244, 4250, 4254, 4267, 4289, 4291, 4315, 4319, 4368, 4370, 4373, 4379, 4405,
4420, 4444, 4475, 4509, 4514, 4537, 4562, 4568, 4594, 4611, 4642, 4643, 4662,
4683, 4726, 4779, 4789, 4876, 4877, 4878, 4903, 4910, 4948, 4956, 4990, 4998,
5003, 5012, 5086, 5102, 5109, 5141, 5166, 5185, 5212, 5246, 5274]
d
[1653, 1274, 683, 4237, 7291, 7022, 499, 6566, 956, 12361, 2273, 4740, 7231,
5693, 17535, 5851, 3754, 5734, 5748, 1871, 3036, 8860, 6743, 10257, 4376, 6886,
2355, 920, 3985, 7539, 3220, 1430, 9802, 8269, 17657, 9754, 2953, 18576, 5789,
6649, 13608, 20906, 1256, 5419, 9168, 2149, 9160, 3764, 2085, 7151, 8404, 4097,
8132, 9647, 5384, 3132, 7837, 2736, 3852, 20142, 3259, 5922, 3564, 8043, 6688,
6967, 6871, 4696, 14848, 2245, 6409, 7160, 6515, 4009, 5834, 2989, 4071, 3748,
3628, 2634, 8579, 7688, 14355, 10790, 8446, 9242, 17217, 2769, 5092, 14926,
12880, 9399, 4836, 5952, 10726, 5712, 3161, 24580, 6862, 6771, 5007, 5309,
25457, 3713, 14273, 11156, 15824, 20413, 10546, 6995, 5869, 6110, 5992, 5980,
10857, 14021, 18216, 5134, 9709, 3772, 6917, 4419, 5122, 13725, 6241, 10165,
9315, 8992, 6508, 4788, 21040, 8278, 17517, 14468, 18058, 14332, 5059, 10607,
24110, 15952, 29200, 4824, 7139, 9630, 6140, 12606, 4968, 7599, 10169, 5389,
6909, 5582, 12165, 5693, 12741, 10125, 4938, 9310, 12376, 5695, 10949, 8007,
9125, 6158, 9109, 11212, 4913, 11588, 4675, 8023, 6372, 13904, 6884, 4960, 6218,
6002, 11671, 5589, 5503, 9818, 5247, 19407, 5599, 5839, 5414, 6941, 11527, 6971,
14055, 5395, 6641, 5544, 9417, 6893, 9865, 5285, 9039, 6887, 14091, 5398]
p
[73, 13, 6, 1, 29, 28, 72, 76, 86, 48, 94, 18, 32, 24, 33, 63, 11, 16, 69, 40,
38, 20, 45, 78, 61, 30, 80, 16, 57, 50, 2, 32, 97, 86, 27, 35, 76, 51, 66, 54,
71, 42, 35, 41, 23, 64, 80, 57, 29, 5, 8, 67, 43, 97, 8, 25, 61, 46, 84, 50, 54,
30, 77, 89, 77, 34, 3, 89, 43, 82, 52, 63, 24, 94, 99, 88, 19, 91, 91, 17, 78,
91, 33, 71, 5, 29, 85, 36, 29, 70, 55, 65, 74, 85, 57, 47, 39, 36, 15, 83, 93,
91, 53, 83, 42, 36, 68, 20, 80, 81, 33, 55, 34, 35, 34, 29, 88, 61, 9, 11, 76,
44, 78, 62, 99, 80, 86, 77, 64, 68, 45, 61, 28, 42, 85, 85, 63, 92, 1, 69, 14,
6, 60, 4, 53, 49, 12, 66, 75, 69, 38, 28, 85, 28, 73, 20, 96, 58, 32, 18, 18,
71, 72, 49, 26, 32, 40, 89, 36, 43, 69, 26, 58, 11, 48, 35, 44, 35, 69, 67, 81,
```

78, 18, 24, 61, 54, 41, 11, 55, 71, 56, 96, 90, 5, 86, 97, 47, 12, 10, 27]

4.1 CP model

[12]:
```
from docplex.cp.model import CpoModel
from docplex.cp.config import context
import sys

# Create model
m = CpoModel()

# - add variables
tasks = [m.interval_var(name="task{:d}".format(i), optional=False, size=p[i])␣
 ↪for i in range(n)]
seq = m.sequence_var(tasks, name='seq')

# - set objective
m.add(m.minimize(m.max([m.end_of(tasks[i]) for i in range(n)]) ) )  # minimize␣
 ↪C_max

# - add constraints
for i in range(n):
    m.add(m.start_of(tasks[i]) >= r[i])  # release time
    m.add(m.end_of(tasks[i]) <= d[i])    # deadline

m.add(m.no_overlap(seq))  # one task executed at one time

# Solve the model
msol = m.solve(TimeLimit=10, LogVerbosity="Normal", LogPeriod=1, Workers=1)


# Print the solution
print()
if msol.is_solution():
    starts = [msol.get_value(tasks[i])[0] for i in range(n)]
    print(*starts, sep="\n")
else:
    print("No solution found.")

print("Done")
```

```
0
237
145
344
1083
```

991
73
624
151
8721
250
345
1051
374
10165
398
363
461
477
546
586
1162
761
7796
700
902
806
886
932
1112
989
1019
7371
6370
8694
1182
1316
9891
1519
1660
8852
9079
1217
1478
6910
1252
6830
1421
1392
5838
6517
1585
6327

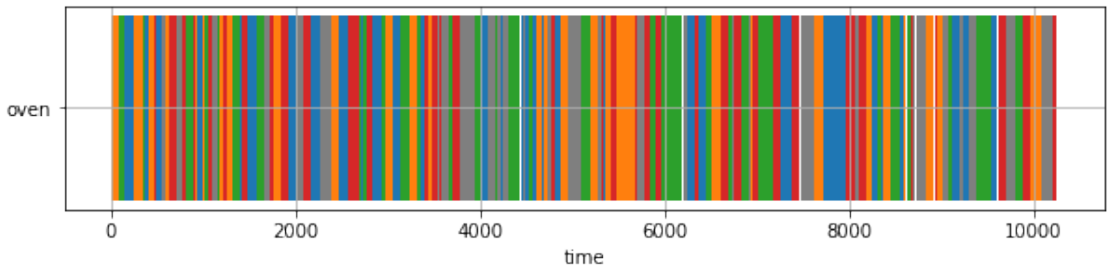
7265
1652
1760
6063
1714
1916
9192
1785
2735
1839
6238
3582
5652
4875
2082
9567
2000
3434
5843
3558
2171
2265
2381
2687
2560
2469
2364
6530
5972
8661
9121
6525
6933
9382
2651
2706
9733
8193
7106
2765
2839
9022
3230
2924
8923
4229
4081
2963

3139
10198
3056
8066
9970
9467
8046
7874
5697
3277
3310
3400
3365
9699
8413
9803
3486
7362
3547
5522
3659
3703
8248
3849
7641
7020
6608
4011
3781
8977
6456
9942
8546
9297
10006
3948
7954
10164
8769
8838
4075
5778
7261
4176
8364
4164
5906
7721

4244
5484
4313
8108
4437
10091
7621
4341
6962
9535
4465
8959
6124
6758
4483
6732
8588
4532
9610
4572
6195
4608
8442
5409
4677
4723
4688
8310
4771
4806
7468
4878
8468
5311
5329
4959
5598
8620
5686
9242
5020
5353
5091
7171
5479
7535
5187
6685

```
5467
8354
5284
Done
```

[13]:
```
plot_solution(starts, p)
```



4.2 ILP model

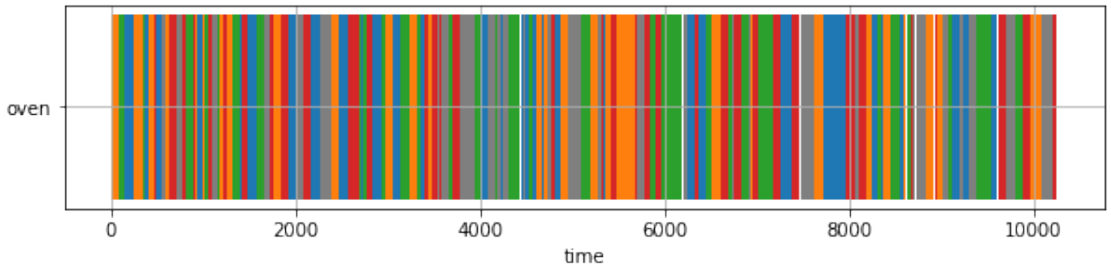
[14]:
```
import gurobipy as g
m = g.Model()

# - add variables
s = m.addVars(n, vtype=g.GRB.CONTINUOUS, lb=0)
x = {}
for i in range(n):
    for j in range(i + 1, n):
        x[i, j] = m.addVar(vtype=g.GRB.BINARY)

Cmax = m.addVar(vtype=g.GRB.CONTINUOUS, obj=1)

# - add constraints
for i in range(n):
    m.addConstr(s[i] + p[i] <= Cmax)
    m.addConstr(s[i] >= r[i])
    m.addConstr(s[i] + p[i] <= d[i])

M = max(d)
for i in range(n):
    for j in range(i + 1, n):
        m.addConstr(s[i] + p[i] <= s[j] + M*(1-x[i, j]))
        m.addConstr(s[j] + p[j] <= s[i] + M*x[i, j])

# call the solver ------------------------------------------------
m.optimize()
```

```python
print()
if m.SolCount > 0:
    starts = [s[i].X for i in range(n)]
else:
    print("No solution was found.")

print("Done")
```

```
Gurobi Optimizer version 9.1.0 build v9.1.0rc0 (linux64)
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads
Optimize a model with 40400 rows, 20101 columns and 120200 nonzeros
Model fingerprint: 0x74609c7a
Variable types: 201 continuous, 19900 integer (19900 binary)
Coefficient statistics:
  Matrix range     [1e+00, 3e+04]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 3e+04]
Presolve removed 9123 rows and 4458 columns
Presolve time: 0.19s
Presolved: 31277 rows, 15643 columns, 93438 nonzeros
Variable types: 201 continuous, 15442 integer (15442 binary)

Root simplex log...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
   20011    5.3531148e+03   1.228794e+03   0.000000e+00      5s
   25781    5.3530000e+03   0.000000e+00   0.000000e+00      8s

Root relaxation: objective 5.353000e+03, 25781 iterations, 7.27 seconds
Total elapsed time = 10.40s

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 5439.00000    0  516          - 5439.00000      -     -   12s
     0     0 5439.00000    0  756          - 5439.00000      -     -   14s
     0     0 5439.00000    0  683          - 5439.00000      -     -   15s
     0     0 5439.00000    0  365          - 5439.00000      -     -   16s
     0     0 5439.00000    0  365          - 5439.00000      -     -   17s
     0     0 5439.00000    0  347          - 5439.00000      -     -   19s
     0     0 5439.00000    0  399          - 5439.00000      -     -   19s
     0     0 5439.00000    0  323          - 5439.00000      -     -   22s
     0     0 5439.00000    0  346          - 5439.00000      -     -   22s
     0     0 5439.00000    0  327          - 5439.00000      -     -   24s
     0     0 5439.00000    0  332          - 5439.00000      -     -   24s
```

```
     0     0 5439.00000    0  329          - 5439.00000      -     -   26s
     0     0 5439.00000    0  352          - 5439.00000      -     -   26s
     0     0 5439.00000    0  307          - 5439.00000      -     -   29s
     0     0 5439.00000    0  307          - 5439.00000      -     -   29s
     0     2 5439.00000    0  307          - 5439.00000      -     -   33s
   203   161 infeasible   65               - 5442.86736      -  58.8   35s
   732   538 5480.00000   14  456          - 5445.29630      -  62.3   42s
   735   540 5568.00000   45  476          - 5445.29630      -  62.1   45s
   740   544 5475.00000    6  434          - 5445.29630      -  61.7   52s
   741   544 5529.22963   21  434          - 5445.29630      -  61.6   55s
   744   549 5445.29630   17  464          - 5445.29630      -  43.3   60s
   777   577 5627.00000   25  371          - 5449.00000      -  47.3   65s
   877   644 5836.00000   54  351          - 5449.00000      -  51.1   70s
   999   731 5938.00000   80  444          - 5449.00000      -  55.6   75s
  1150   877 5897.00000  110  439          - 5449.00000      -  54.2   80s

Cutting planes:
  Cover: 7
  Implied bound: 5
  MIR: 5
  Relax-and-lift: 8

Explored 1233 nodes (218830 simplex iterations) in 80.55 seconds
Thread count was 4 (of 4 available processors)

Solution count 0

Solve interrupted
Best objective -, best bound 5.449000000000e+03, gap -

No solution was found.
Done
```

[15]:
```
plot_solution(starts, p)
```



5 TSP

[22]:
```
import math
from collections import namedtuple
import gurobipy as g

Point = namedtuple("Point", ['x', 'y'])

def length(point1, point2):
    return int(round(math.sqrt((point1.x - point2.x)**2 + (point1.y - point2.
 ↪y)**2)))  # CP works with int only

class TSP:
    def __init__(self):
        D = None  # distance matrix
        points = None  # vertices

    def load_instance(self, path):
        input_data_file = open(path, 'r')
        input_data = ''.join(input_data_file.readlines())

        # parse the input
        lines = input_data.split('\n')
        nodeCount = int(lines[0])

        points = []
        for i in range(1, nodeCount+1):
            parts = lines[i].split()
            points.append(Point(float(parts[0]), float(parts[1])))

        # distance matrix
        D = [[0 for _ in range(nodeCount+1)] for _ in range(nodeCount+1)]  #␣
 ↪Add dummy vertex last
        for i in range(nodeCount):
            for j in range(nodeCount):
                D[i][j] = length(points[i], points[j])

        # the last vertex is the same as the first one
        for i in range(nodeCount):
            D[i][-1] = D[i][0]
            D[-1][i] = D[0][i]


        self.D = D
        self.points = points

        return self
```

[23]:
```
instances = [
        {"inst": TSP().load_instance("./tsp_data/tsp_5_1"),
         "init": [0, 1, 2, 4, 3]},
        {"inst": TSP().load_instance("./tsp_data/tsp_51_1"),
         "init": [0, 5, 2, 28, 10, 9, 45, 3, 46, 8, 4, 35, 13, 7, 19, 40, 18,␣
↪11, 42, 37, 20, 25, 1, 31, 22, 48, 32, 17, 49, 39, 50, 38, 15, 44, 14, 16,␣
↪29, 43, 21, 30, 12, 23, 34, 24, 41, 27, 36, 6, 26, 47, 33]},
        {"inst": TSP().load_instance("./tsp_data/tsp_70_1"),
         "init": [0, 35, 50, 11, 57, 2, 56, 27, 21, 49, 58, 53, 41, 36, 38, 52,␣
↪6, 5, 7, 51, 55, 68, 46, 67, 24, 16, 44, 39, 22, 1, 14, 15, 20, 29, 28, 45,␣
↪12, 31, 18, 26, 3, 59, 9, 25, 4, 10, 61, 43, 32, 8, 64, 54, 48, 62, 13, 19,␣
↪60, 42, 37, 66, 40, 17, 30, 23, 69, 33, 65, 34, 47, 63]},
        {"inst": TSP().load_instance("./tsp_data/tsp_100_1"),
         "init": [0, 6, 69, 61, 76, 35, 84, 11, 9, 26, 72, 47, 40, 94, 81, 60,␣
↪64, 66, 8, 23, 70, 59, 33, 67, 43, 37, 65, 71, 19, 15, 75, 14, 53, 46, 5,␣
↪29, 80, 38, 91, 57, 41, 50, 12, 55, 98, 39, 24, 68, 2, 28, 73, 87, 48, 85,␣
↪21, 96, 42, 77, 16, 7, 10, 74, 30, 18, 17, 34, 22, 99, 93, 51, 3, 89, 13,␣
↪31, 44, 62, 25, 82, 86, 54, 1, 27, 45, 88, 79, 97, 49, 90, 20, 63, 52, 92,␣
↪95, 78, 83, 32, 4, 56, 58, 36]},
        {"inst": TSP().load_instance("./tsp_data/tsp_200_1"),
         "init": [0, 103, 62, 192, 5, 48, 89, 148, 117, 9, 128, 83, 136, 23,␣
↪37, 108, 177, 181, 98, 106, 35, 160, 125, 131, 123, 58, 73, 20, 145, 71,␣
↪111, 46, 97, 22, 114, 112, 178, 59, 61, 163, 119, 154, 141, 34, 85, 26, 11,␣
↪19, 146, 130, 166, 76, 164, 179, 60, 24, 80, 101, 134, 68, 167, 129, 188,␣
↪158, 102, 172, 88, 168, 41, 30, 79, 55, 199, 132, 144, 96, 180, 196, 3, 64,␣
↪65, 195, 25, 186, 151, 110, 183, 147, 69, 21, 15, 87, 143, 162, 93, 150,␣
↪115, 17, 78, 52, 165, 18, 191, 198, 118, 109, 74, 135, 156, 173, 7, 113, 91,␣
↪159, 57, 176, 50, 86, 56, 6, 8, 105, 153, 174, 82, 54, 107, 121, 33, 28, 45,␣
↪116, 124, 133, 189, 42, 2, 13, 197, 157, 40, 70, 99, 187, 47, 127, 138, 137,␣
↪170, 29, 171, 182, 161, 84, 67, 72, 122, 49, 43, 169, 175, 190, 193, 194,␣
↪149, 38, 185, 95, 155, 51, 77, 104, 4, 142, 36, 32, 75, 12, 94, 81, 1, 63,␣
↪39, 120, 53, 140, 66, 27, 92, 126, 90, 44, 184, 31, 100, 152, 14, 16, 10,␣
↪139]}
]
```

[24]:
```
inst_id = 4
inst = instances[inst_id]["inst"]
init_order = instances[inst_id]["init"]
```

[25]:
```
INITIALIZE = False
```

5.1 ILP

[26]:
```
nodeCount = len(inst.points)
points = inst.points

# Create model
m = g.Model("tsp")

# - add variables
x = m.addVars(nodeCount,nodeCount, vtype=g.GRB.BINARY, name="x")
u = m.addVars(nodeCount, vtype=g.GRB.INTEGER, lb=0, name="u")

# - set objective
obj = g.quicksum(g.quicksum(inst.D[i][j]*x[i,j] for j in range(nodeCount)) for␣
 ↪i in range(nodeCount))
m.setObjective(obj, g.GRB.MINIMIZE)

# - add constraints
m.addConstrs((1 == g.quicksum(x[i,j] for j in range(nodeCount)) for i in␣
 ↪range(nodeCount)))
m.addConstrs((1 == g.quicksum(x[j,i] for j in range(nodeCount)) for i in␣
 ↪range(nodeCount)))

for i in range(1, nodeCount):
    for j in range(1, nodeCount):
        m.addConstr(u[i]-u[j]+1 <= nodeCount*(1-x[i,j]))

# Initialization
if INITIALIZE:
    for i, order in enumerate(init_order):
        u[order].start = i

m.Params.TimeLimit = 10
m.optimize()


# Print the solution
print()
if m.SolCount > 0:
    obj = m.objVal
    print("Objective {}".format(obj))

    order = [u[i].X for i in range(nodeCount)]
    indices = range(nodeCount)
    s = sorted(zip(order,indices), key=lambda x: x[0])
    print([x[1] for x in s])
else:
```

```
    print("No solution was found.")

print("Done")
```

```
Changed value of parameter TimeLimit to 10.0
   Prev: inf  Min: 0.0  Max: inf  Default: inf
Gurobi Optimizer version 9.1.0 build v9.1.0rc0 (linux64)
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads
Optimize a model with 40001 rows, 40200 columns and 198405 nonzeros
Model fingerprint: 0xceefc9da
Variable types: 0 continuous, 40200 integer (40000 binary)
Coefficient statistics:
  Matrix range     [1e+00, 2e+02]
  Objective range  [1e+01, 4e+03]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 2e+02]
Presolve removed 199 rows and 200 columns
Presolve time: 0.32s
Presolved: 39802 rows, 40000 columns, 197808 nonzeros
Variable types: 0 continuous, 40000 integer (39801 binary)

Deterministic concurrent LP optimizer: primal and dual simplex
Showing first log only…

Concurrent spin time: 0.01s

Solved with dual simplex

Root relaxation: objective 2.312896e+04, 677 iterations, 0.22 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 23128.9600    0  411          - 23128.9600      -     -    3s

Explored 1 nodes (772 simplex iterations) in 10.30 seconds
Thread count was 4 (of 4 available processors)

Solution count 0

Time limit reached
Best objective -, best bound 2.312900000000e+04, gap -

No solution was found.
Done
```

5.2 CP

[27]:
```
from docplex.cp.model import CpoModel
from docplex.cp.config import context
import sys

node_count = len(inst.points)
points = inst.points

# Create model
m = CpoModel()

# - add variables
cities = [m.interval_var(name="city{:d}".format(i), optional=False, size=1) for␣
 ↪i in range(node_count+1)]
seq = m.sequence_var(cities, name='seq', types=([i for i in range(node_count)]␣
 ↪+ [0]))

# - set objective
m.add(m.minimize(m.max([m.end_of(cities[i]) for i in range(len(cities))]) -␣
 ↪len(cities)))

# - add constraints
m.add(m.first(seq, cities[0]))  # start from city 0
m.add(m.last(seq, cities[-1]))  # repeat the same city last
m.add(m.no_overlap(seq, inst.D, True))

# Solve the model
msol = m.solve(TimeLimit=10, LogVerbosity="Verbose", LogPeriod=1, Workers=1)

# Print the solution
print()
if msol.is_solution():

    ovals = msol.get_objective_values()
    print("Objective {}".format(ovals[0]))

    starts = [msol.get_value(cities[i])[0] for i in range(len(cities)-1)]
    indices = range(len(cities)-1)
    s = sorted(zip(starts,indices), key=lambda x: x[0])

    print([x[1] for x in s])
else:
    print("No solution found.")

print("Done")
```

```
Objective 31962
[0, 103, 62, 192, 5, 48, 89, 148, 117, 9, 128, 83, 136, 23, 37, 108, 177, 181,
98, 106, 125, 35, 160, 131, 123, 58, 73, 20, 145, 71, 111, 97, 22, 114, 112,
178, 59, 61, 163, 119, 154, 141, 34, 85, 26, 11, 19, 146, 130, 166, 76, 164,
179, 60, 24, 80, 101, 134, 68, 167, 129, 188, 158, 102, 172, 88, 168, 41, 30,
79, 55, 199, 132, 144, 96, 180, 196, 3, 64, 65, 195, 25, 186, 151, 110, 183,
147, 69, 21, 15, 87, 143, 162, 93, 150, 115, 17, 78, 52, 165, 18, 191, 198, 118,
109, 74, 135, 156, 173, 7, 113, 91, 159, 57, 176, 50, 86, 56, 6, 8, 105, 153,
174, 82, 54, 107, 121, 33, 28, 45, 116, 124, 133, 189, 42, 2, 13, 197, 157, 40,
70, 99, 187, 47, 127, 138, 137, 170, 29, 171, 182, 161, 84, 67, 72, 122, 49, 43,
169, 175, 190, 193, 194, 149, 38, 185, 95, 155, 51, 77, 104, 4, 142, 36, 32, 75,
12, 94, 81, 1, 63, 39, 120, 53, 140, 66, 27, 92, 126, 90, 44, 184, 31, 100, 152,
14, 16, 10, 139, 46]
Done
```

5.3 Comparison

Best objective found by ILP and CP model under 10s timelimit. (without initialization)

instance	ILP	CP
5	3	3
50	496	441
70	994	710
100	-	22247
200	-	31962

[ ]: