Emparelhamento m?ximo

September 25, 2020

In [1]:
```
# dessa vez a gente vai usar as definições do sage,
# em particular a implementação de grafos.

from sage.all import *
from docplex.mp.model import Model
import itertools as it
```

In [2]:
```
# vamos criar uma função que, dado um grafo,
# retorna um modelo matemático para o problema do emparelhamento máximo

def maximum_matching(G):
    # criamos o modelo
    mdl = Model()
    # criamos as variáveis
    # as arestas têm uma terceira componente, que pode ser usada como peso,
    # e como não precisamos de peso, vamos ignorá-la
    x = {(u, v) : mdl.binary_var() for (u, v, _) in G.edges()}
    # uma restrição para cada vértice
    for v in G.vertices():
        mdl.add_constraint(
            sum(x[a, b] for (a, b, _) in G.edges_incident(v)) <= 1)
    # por fim, a função-objetivo de maximização
    mdl.maximize(sum(x[u, v] for (u, v, _) in G.edges()))
    # retornamos o modelo pronto, junto com suas variáveis
    return mdl, x
```

In [3]:
```
# agora podemos criar um grafo e testar nosso modelo
# isto cria um grafo aleatório com 100 vértices,
# e chance 0.5 de ocorrência de uma aresta entre cada par de vértices
G = graphs.random.RandomGNP(100, 0.5)
mdl, x = maximum_matching(G)
```

In [4]:
```
# resolvemos o modelo
sol = mdl.solve(log_output=True)
sol.display()
```

```
Version identifier: 12.10.0.0 | 2019-11-26 | 843d4de
CPXPARAM_Read_DataCheck                          1
```

```
CPXPARAM_RandomSeed                              201903125
Found incumbent of value 0.000000 after 0.00 sec. (0.06 ticks)
Tried aggregator 1 time.
Reduced MIP has 100 rows, 2497 columns, and 4994 nonzeros.
Reduced MIP has 2497 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.02 sec. (5.76 ticks)
Tried aggregator 1 time.
Detecting symmetries...
Reduced MIP has 100 rows, 2497 columns, and 4994 nonzeros.
Reduced MIP has 2497 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.02 sec. (6.32 ticks)
Probing time = 0.01 sec. (3.43 ticks)
Clique table members: 100.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 4 threads.
Root relaxation solution time = 0.02 sec. (7.56 ticks)

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                           0.0000     2497.0000             ---
*     0+    0                          49.0000     2497.0000             ---
*     0     0      integral     0       50.0000       50.0000      456    0.00%
Elapsed time = 0.10 sec. (26.06 ticks, tree = 0.00 MB, solutions = 3)

Root node processing (before b&c):
  Real time             =    0.10 sec. (26.17 ticks)
Parallel b&c, 4 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.10 sec. (26.17 ticks)
solution for: docplex_model1
objective: 50
x8 = 1
x83 = 1
x130 = 1
x166 = 1
x231 = 1
x298 = 1
x345 = 1
x382 = 1
x497 = 1
x520 = 1
x584 = 1
x601 = 1
```

```
x654 = 1
x725 = 1
x804 = 1
x814 = 1
x866 = 1
x903 = 1
x944 = 1
x980 = 1
x1017 = 1
x1075 = 1
x1102 = 1
x1115 = 1
x1197 = 1
x1199 = 1
x1235 = 1
x1341 = 1
x1406 = 1
x1451 = 1
x1528 = 1
x1569 = 1
x1587 = 1
x1609 = 1
x1639 = 1
x1668 = 1
x1700 = 1
x1768 = 1
x1849 = 1
x2026 = 1
x2115 = 1
x2195 = 1
x2213 = 1
x2272 = 1
x2282 = 1
x2310 = 1
x2317 = 1
x2333 = 1
x2408 = 1
x2461 = 1
```

In [5]:
```python
# agora vamos tomar o conjunto de arestas escolhidas
# pelo modelo como um emparelhamento máximo
# vamos representar as arestas por conjuntos
# para podermos checar que elas são disjuntas

matching = [set([u, v]) for (u, v, _) in G.edges() if x[u, v].solution_value == 1]
len(matching)
```

Out[5]: 50

```
In [6]: # vamos checar que as arestas selecionadas de fato formam um emparelhamento

        def is_matching(edges):
            return all(e1.isdisjoint(e2)
                       for (e1, e2) in it.combinations(edges, 2))

In [7]: # e agora constatamos que as arestas selecionadas de fato formam um emparelhamento

        is_matching(matching)

Out[7]: True

In [8]: # agora sabemos que de fato o modelo selecionou um emparelhamento,
        # mas terá sido um emparelhamento máximo?

        def is_maximum_matching(G, edges):
            G_edges_as_sets = [set([u, v]) for (u, v, _) in G.edges()]
            cannot_expand = lambda e1: not all(e1.isdisjoint(e2) for e2 in edges)
            cannot_be_expanded = all(cannot_expand(e) for e in G_edges_as_sets)
            return is_matching(edges) and cannot_be_expanded

In [9]: # e verificamos que de fato temos um emparelhamento máximo
        is_maximum_matching(G, matching)

Out[9]: True
```