Set Packing Problem

Alunos: Bruno R. L. Netto
Professor: Abilio Lucena

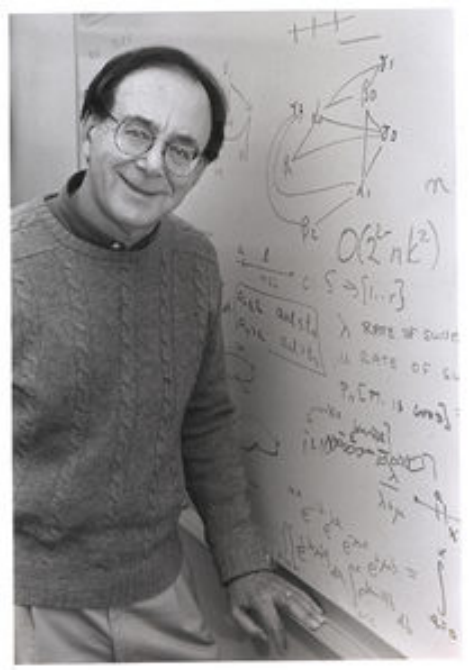
01 de Outubro de 2020

Universidade Federal do Rio de Janeiro - UFRJ

Introdução

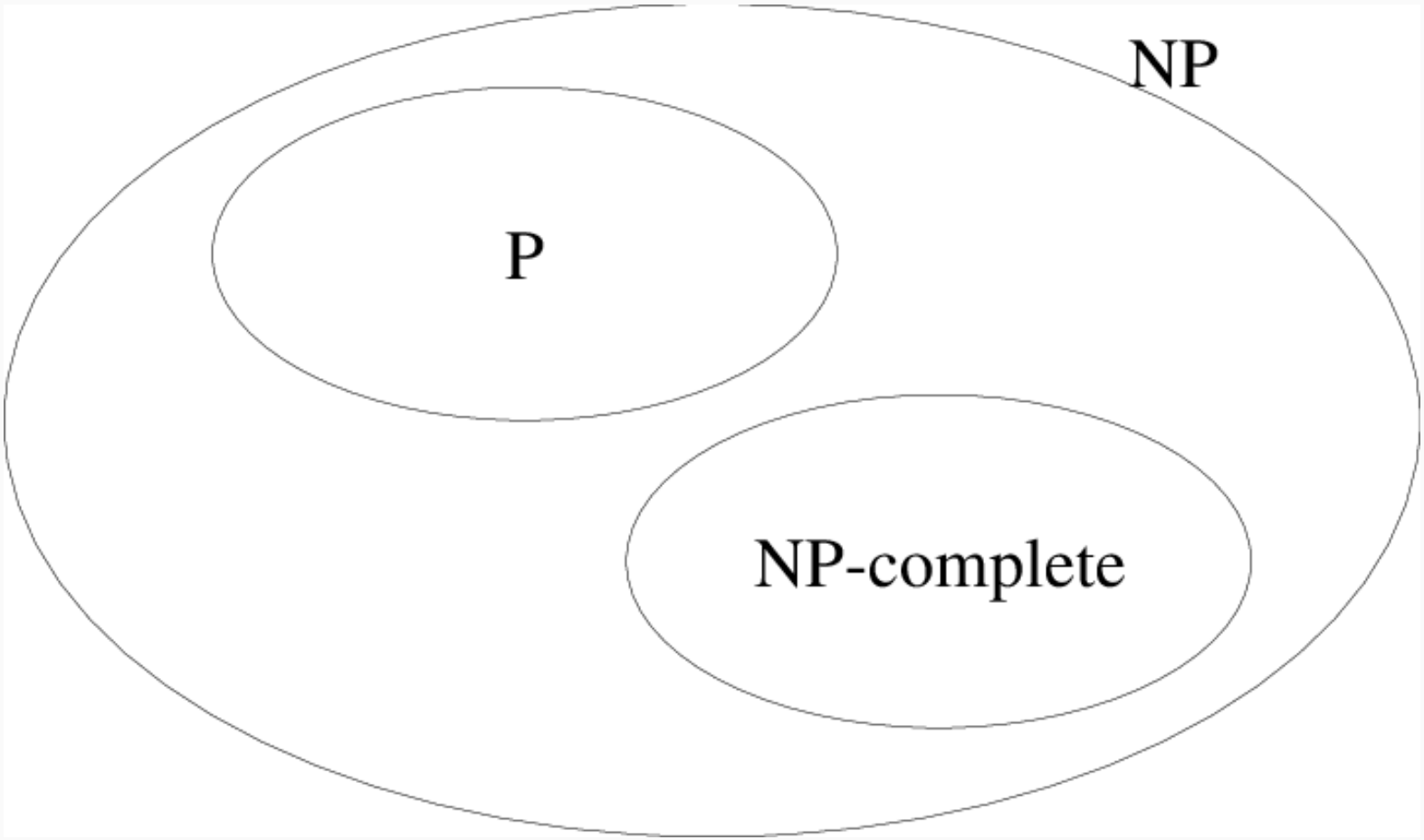
Introdução

(a) Stephen Cook, 1968

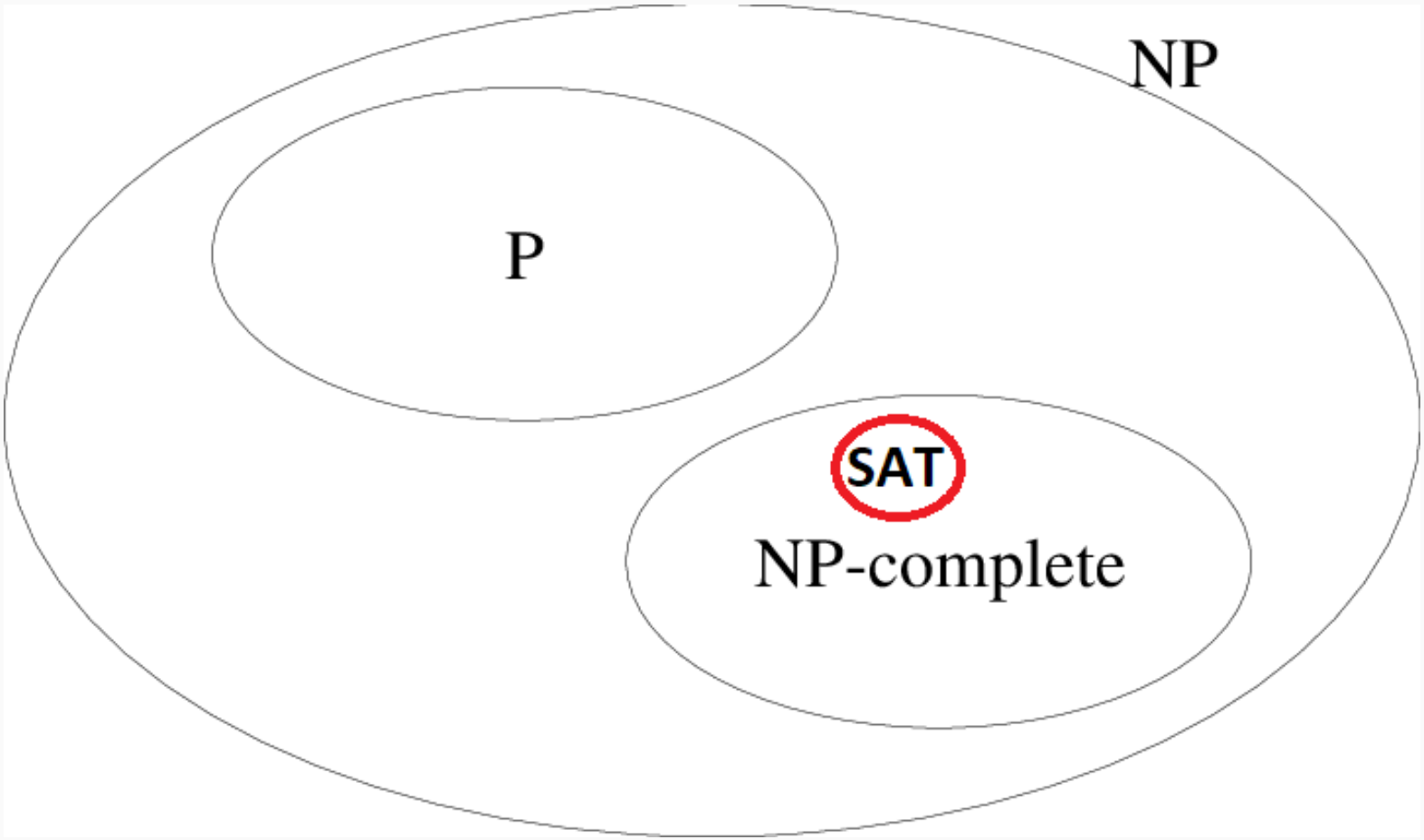


(b) Richard Karp, 2013

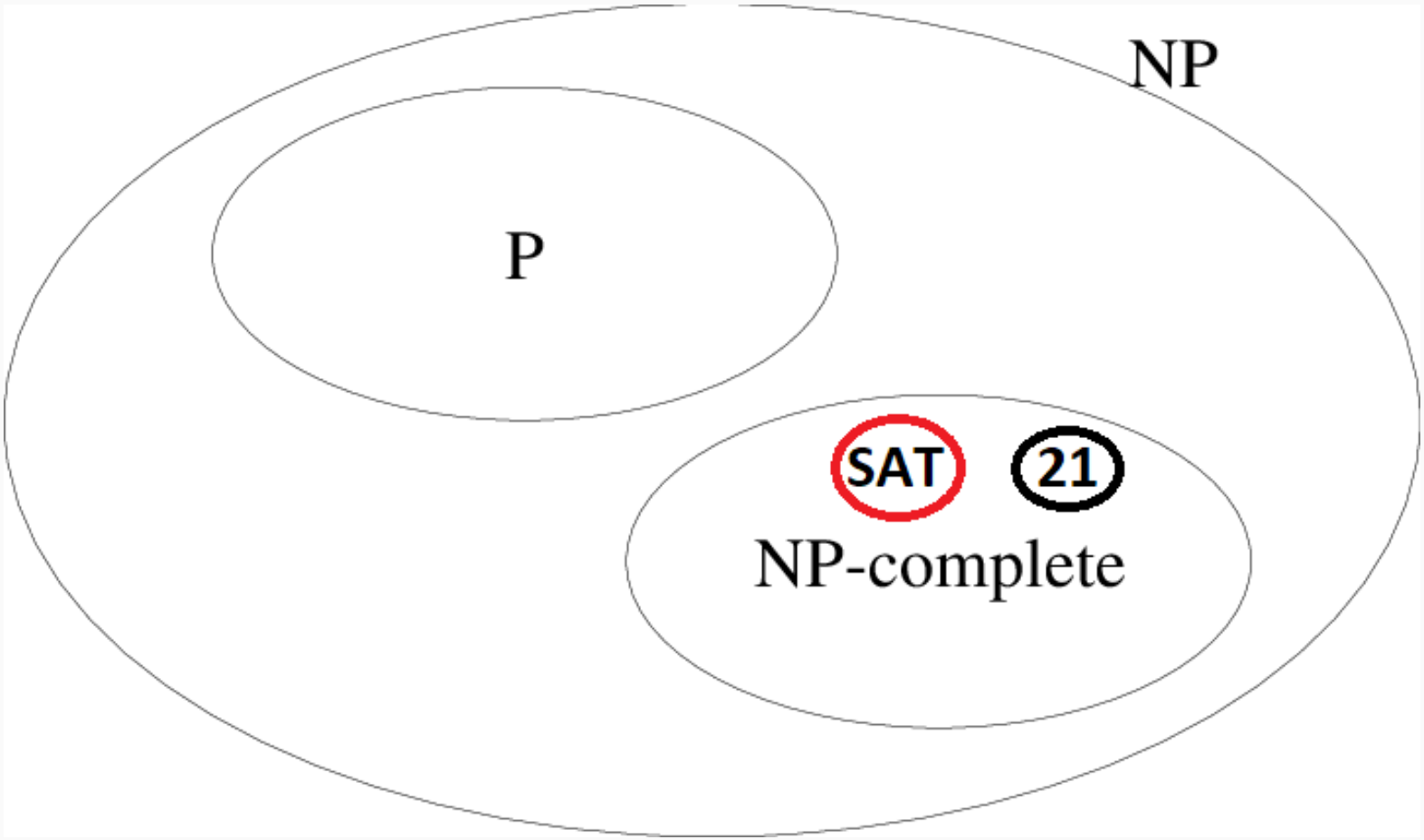
Introdução - Breve história da complexidade



Introdução - Breve história da complexidade



Introdução - Breve história da complexidade



Introdução - Breve história da complexidade

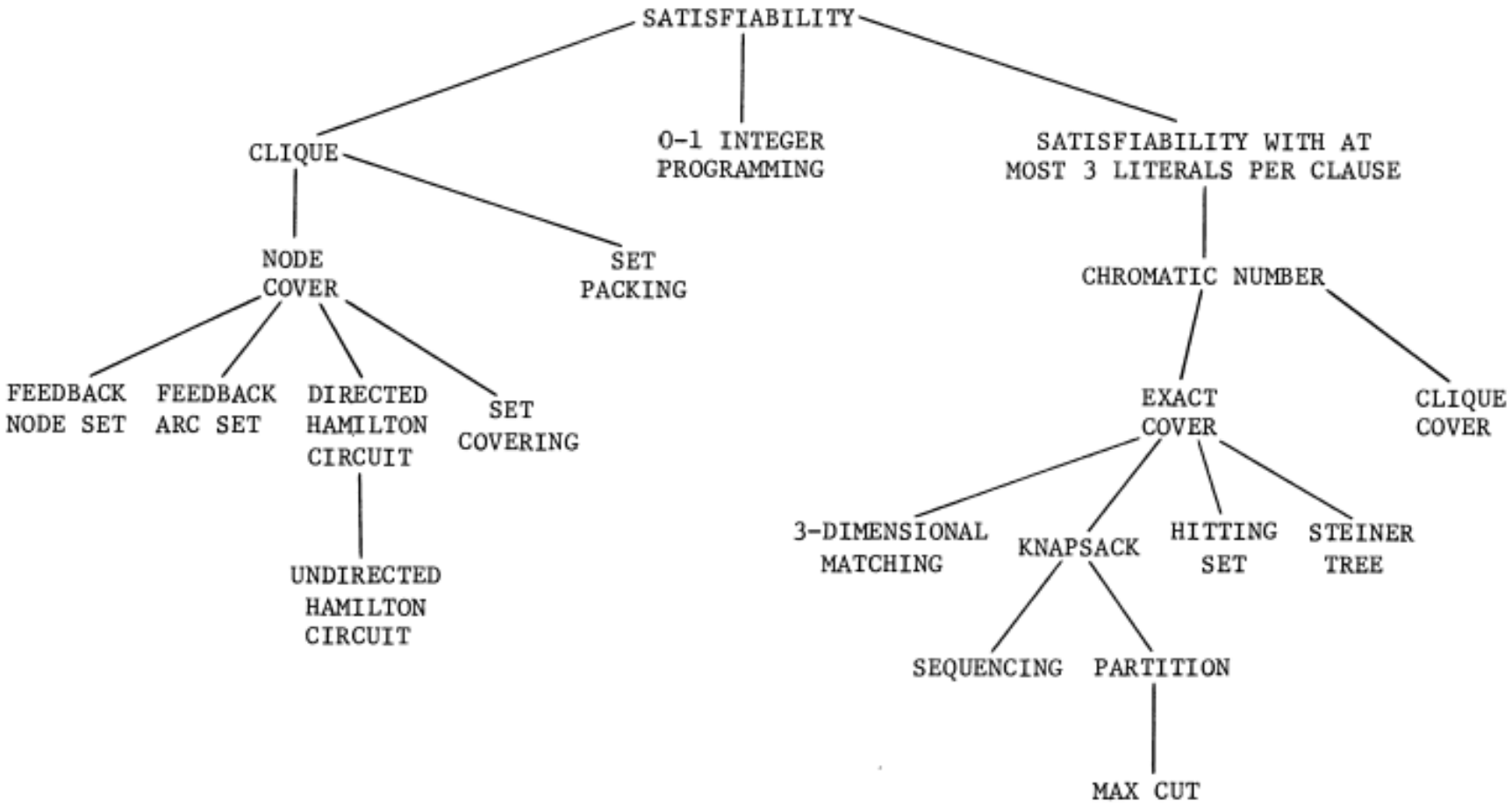
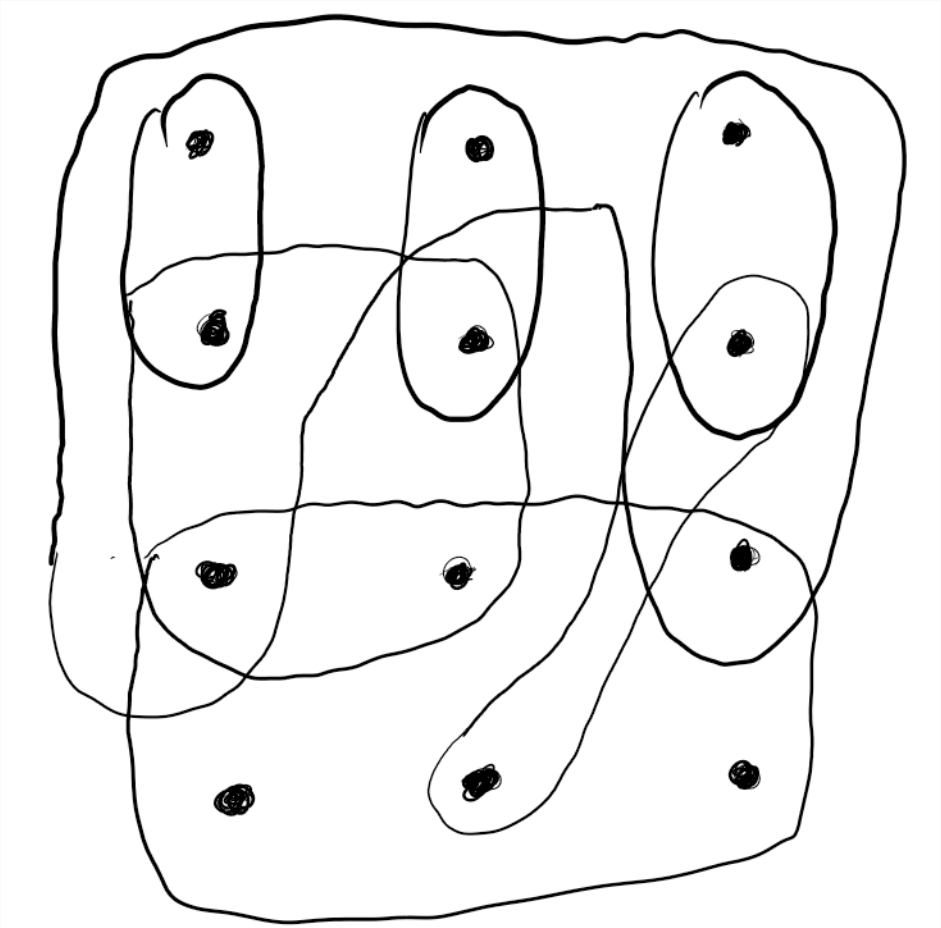


FIGURE 1 - Complete Problems

Introdução - Set Pack Problem(SPP)

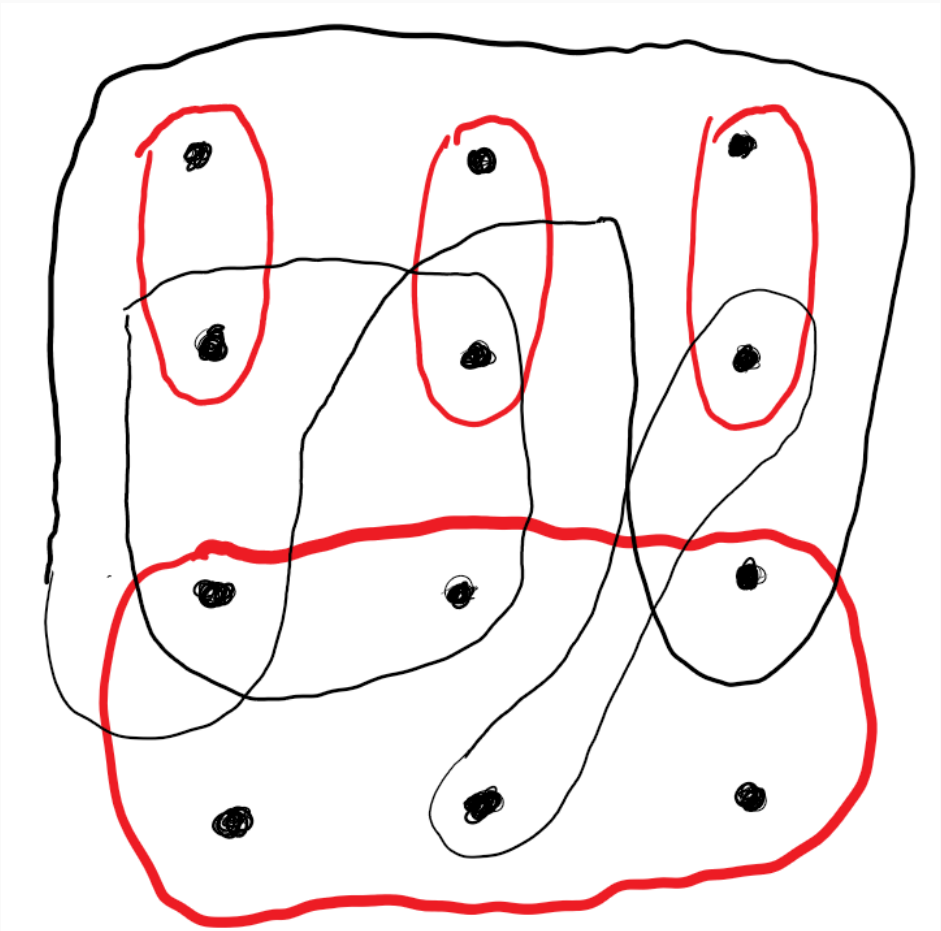
Set Packing - Karp(1972)

Dada uma família de conjuntos $\{S_j\}$, e um inteiro k. Queremos saber se existem k conjuntos mutuamente disjuntos contidos em $\{S_j\}$.



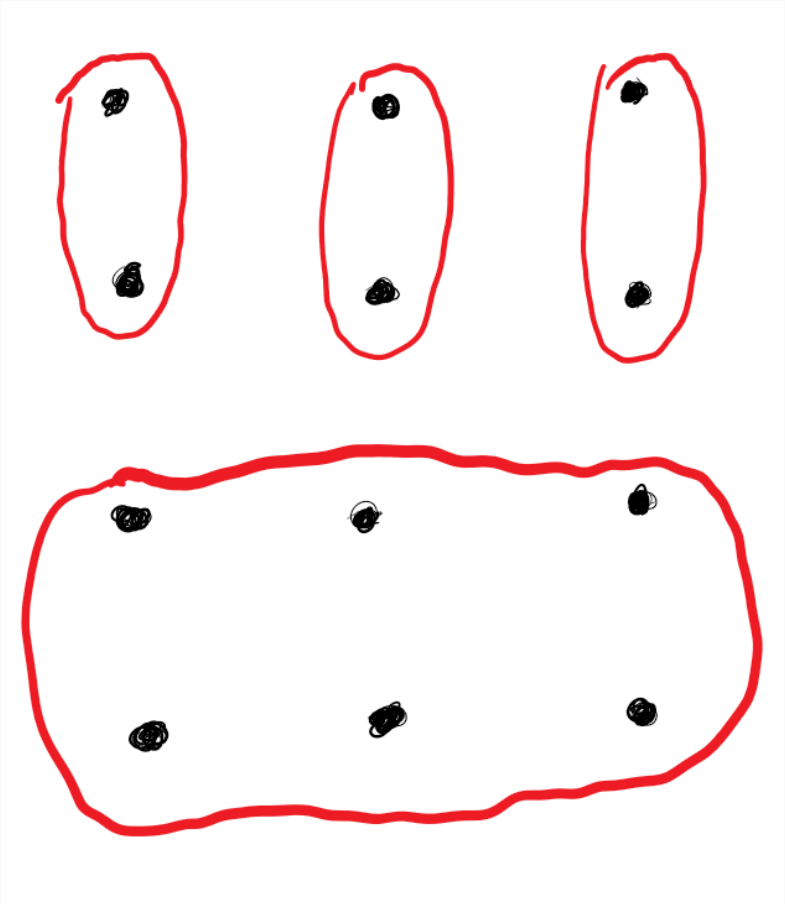
Set Packing - Otimização

Qual o maior inteiro k tal que para uma dada uma família de conjuntos $\{S_j\}$, existam k conjuntos mutuamente disjuntos contidos em $\{S_j\}$.

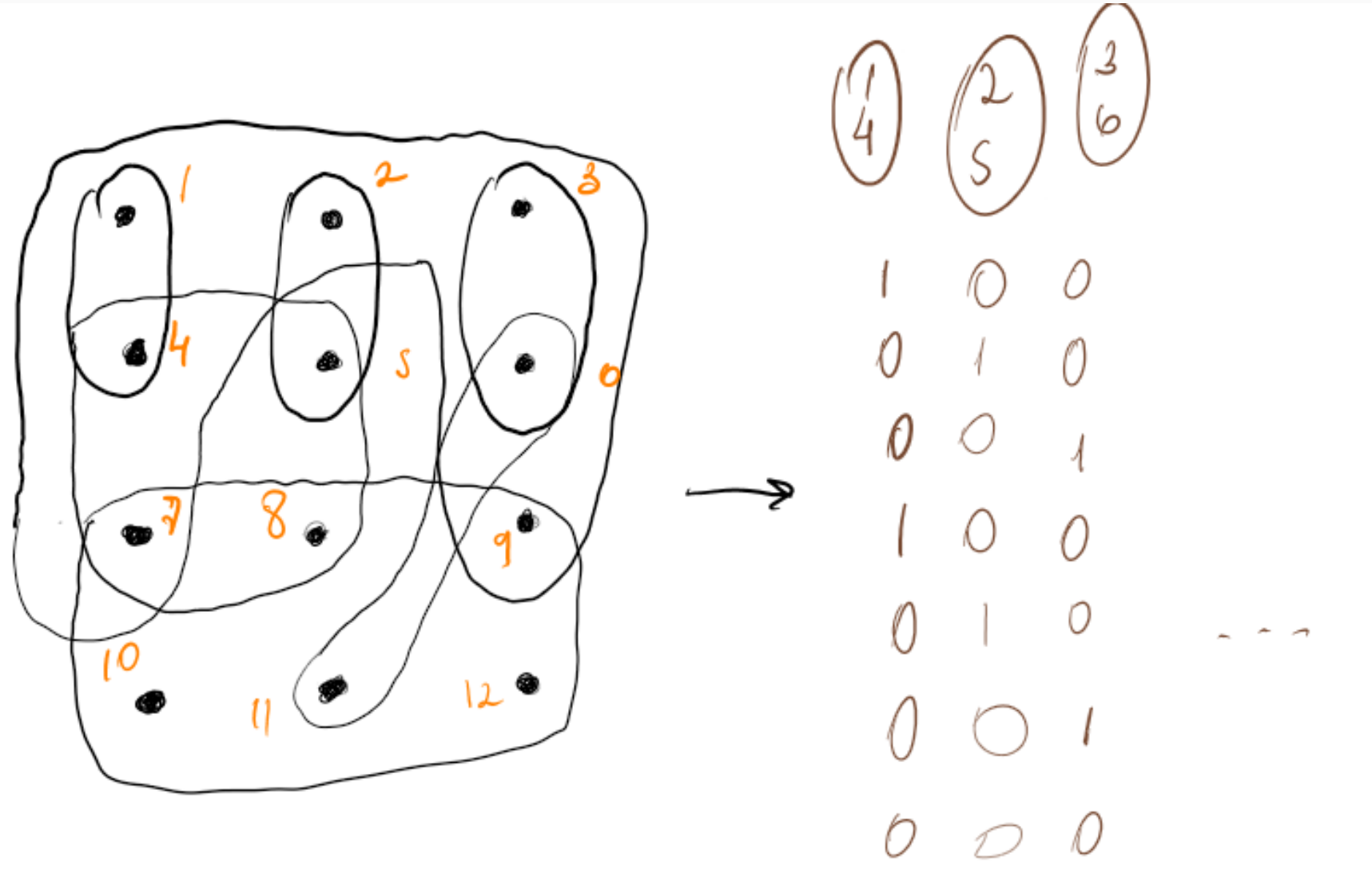


Set Packing - Otimização

Qual o maior inteiro k tal que para uma dada uma família de conjuntos $\{S_j\}$, existam k conjuntos mutuamente disjuntos contidos em $\{S_j\}$.



Introdução - Modelagem do SPP



SPP - Formulação simples

Havendo I subconjuntos e J pontos:

$$\max \sum_{i=1}^{I} x_i$$

$$restrito\ a:$$

$$\sum_{i=1}^{I} a_{i,j} x_j \leq 1, \forall j \in [J]$$

$$x \in \{0,1\}^J$$

SPP - Formulação com peso

Havendo I subconjuntos e J pontos, atribuindo um custo c_i para cada um dos subconjuntos:

$$\max \sum_{i=1}^{I} c_i x_i$$

$$restrito\ a:$$

$$\sum_{i=1}^{I} a_{i,j} x_j \leq 1, \forall j \in [J]$$

$$x \in \{0,1\}^J$$

SPP - Problema Dualizado

Havendo I subconjuntos e J pontos, atribuindo um custo c_i para cada um dos subconjuntos:

$$\mathcal{L}(\mu) = \max \sum_{i=1}^{I} c_i x_i + \sum_{j=1}^{J} \mu_j (1 - \sum_{i=1}^{I} a_{i,j} x_j)$$

$$restrito\ a:$$

$$x \in \{0,1\}^J.$$

Introdução - Pseudocódigo

Data: $A_{I\times J}, c \in \mathbb{R}^I_+$
iter $= 0$;
u $= new\_u$;
while $iter \leq maxiter$ ***or*** $|L_b - U_b| \geq tol$ **do**
    $u \leftarrow u - step * g$;
    $nU_b =$ **Solve** $\mathcal{L}(u)$;
    iter $+= 1$;
    if $nU_b \leq U_b$ **then**
        $U_b \leftarrow nU_b$;
        $L_b \leftarrow updt(x)$;
    end
    $g \leftarrow new\_g$;
    $step \leftarrow new\_step$
end

Algorithm 1: Rotina do Subgradiente

Escolha dos parâmetros

Como escolher:

- Multiplicadores u iniciais.
- Step_size

No artigo Guo et al., "Using a Lagrangian heuristic for a combinatorial auction problem", os autores utilizam um vetor de multiplicadores iniciais dado por:

$$u_j^0 = \frac{\sum_i^{a_{ij}=1} \frac{c_i}{\sum_k^{a_{kj}=1}}}{\sum_p^{a_{pj}=1}} \quad \forall j \in M \qquad (6)$$

Uma breve aplicação

Uma aplicação interesantíssima pode ser encontrada em Lusby et al., "Routing trains through railway junctions: a new set-packing approach".

$$
\begin{aligned}
\mathcal{N}: \quad & \text{maximize} && \rho^T x, && (1)\\
& \text{subject to:} && x_i + x_j \leq 1 \quad \text{for conflicting train} && \\
& && \qquad\qquad \text{paths } i \text{ and } j, && (2)\\
& && x \in \{0,1\}^n. && (3)\\
\\
\mathcal{P}: \quad & \text{maximize} && \rho^T x, && (4)\\
& \text{subject to:} && Tx = \mathbb{1}, && (5)\\
& && Rx \leq \mathbb{1}, && (6)\\
& && x \in \{0,1\}^n. && (7)
\end{aligned}
$$

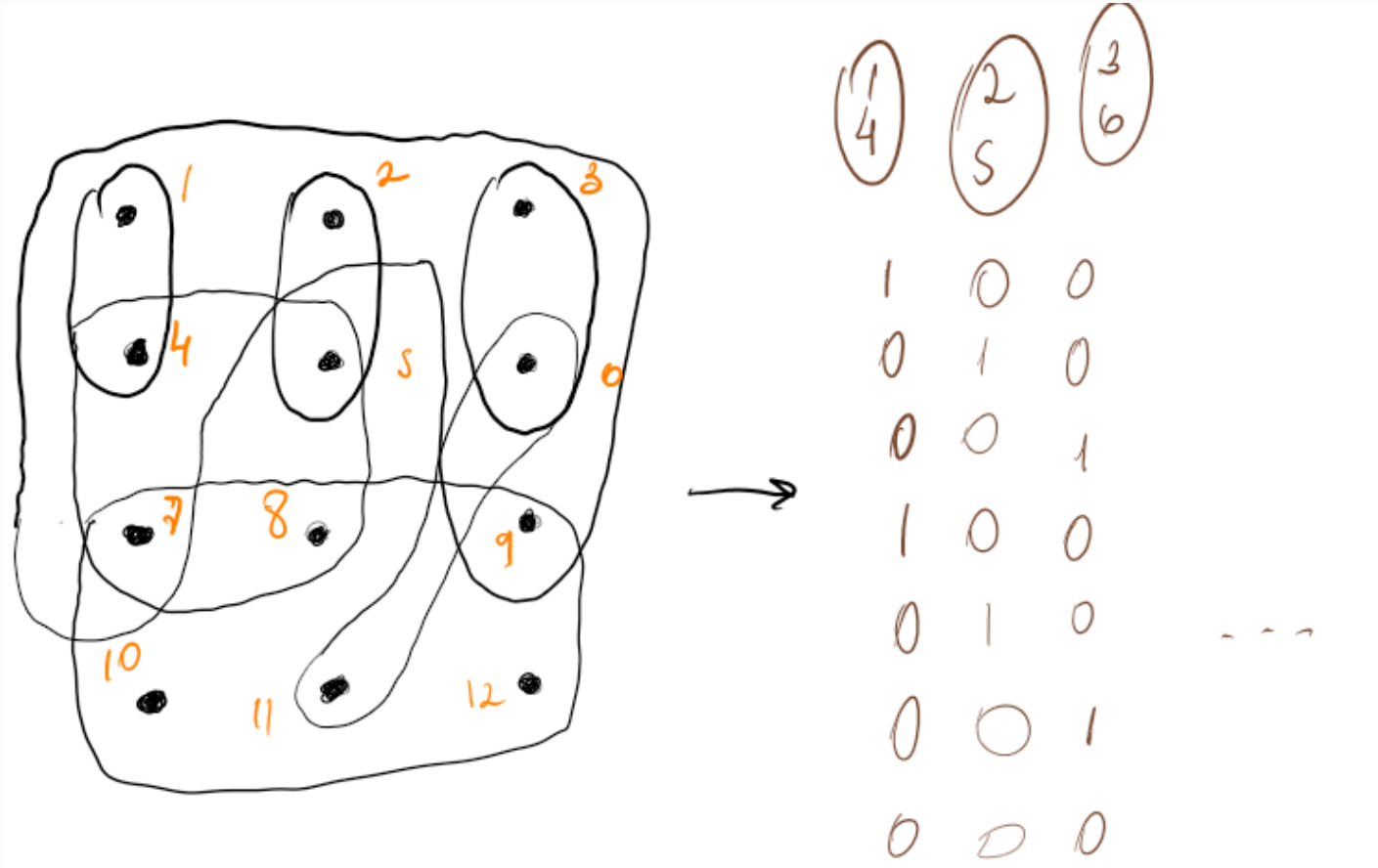
Códigos - Resultados Iniciais

Tipos de Instâncias

Ao longo desse trabalho eu separei as instâncias em 3 tipos:

- Instâncias Minúsculas: onde $I \times J \leq 160$
- Instâncias Médianas: onde $161 \leq I \times J \leq 625$
- Instâncias Colossais: onde $I \times J \geq 625$

Instâncias Mínusculas



Instâncias Mínusculas

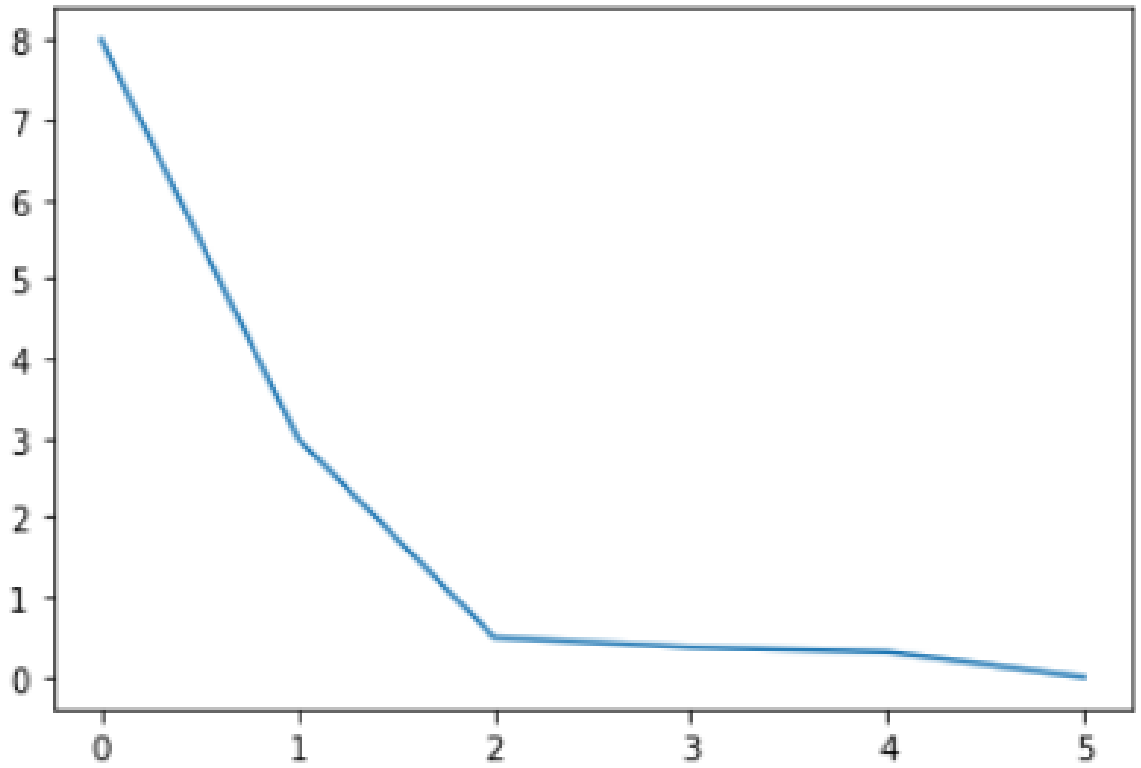
```
12×7 Array{Int64,2}:
 1  0  0  0  1  0  0
 0  1  0  0  1  0  0
 0  0  1  0  1  0  0
 1  0  0  1  1  0  0
 0  1  0  1  0  0  0
 0  0  1  0  1  1  0
 0  0  0  1  1  0  1
 0  0  0  1  0  0  1
 0  0  0  0  1  0  1
 0  0  0  0  0  0  1
 0  0  0  0  0  1  1
 0  0  0  0  0  1  1
```

Instâncias Mínusculas

```
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0   1.2000000e+01   0.000000e+00   0.000000e+00      0s

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       1   7.0000000e+00   0.000000e+00   0.000000e+00      0s

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       2   4.5000000e+00   0.000000e+00   0.000000e+00      0s

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       3   4.3333333e+00   0.000000e+00   0.000000e+00      0s

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       4   4.0333333e+00   0.000000e+00   0.000000e+00      0s

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       5   4.0000000e+00   0.000000e+00   0.000000e+00      0s

Out[160]: ([1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0], 4.0)
```

Instâncias Mínusculas

```
#Gráfico de convergência
plt.plot(zs2 - 4);
```



Instâncias Mínusculas - Casos especiais

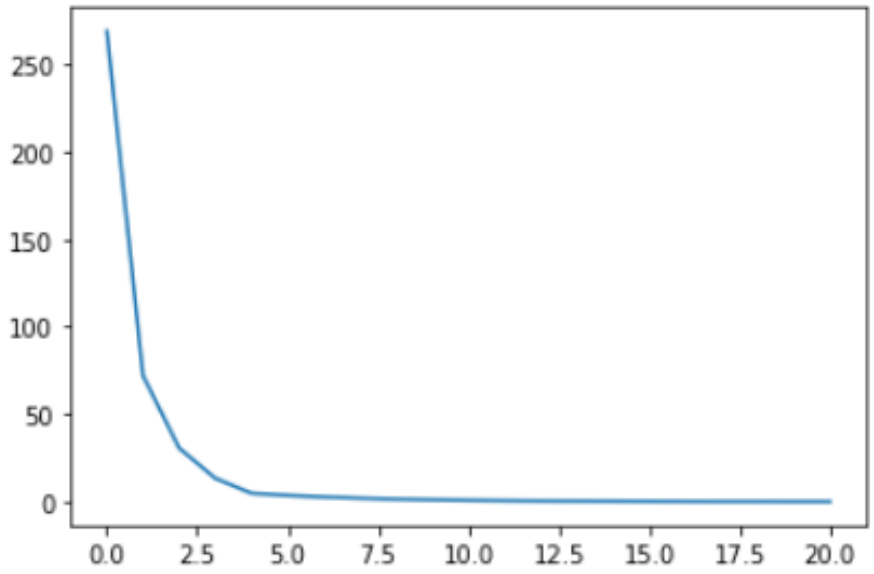
Pela definição temos $I \times J \leq 160$. Podemos distribuir o peso de várias formas:

$$I = J \rightarrow I, J = (13, 12)$$

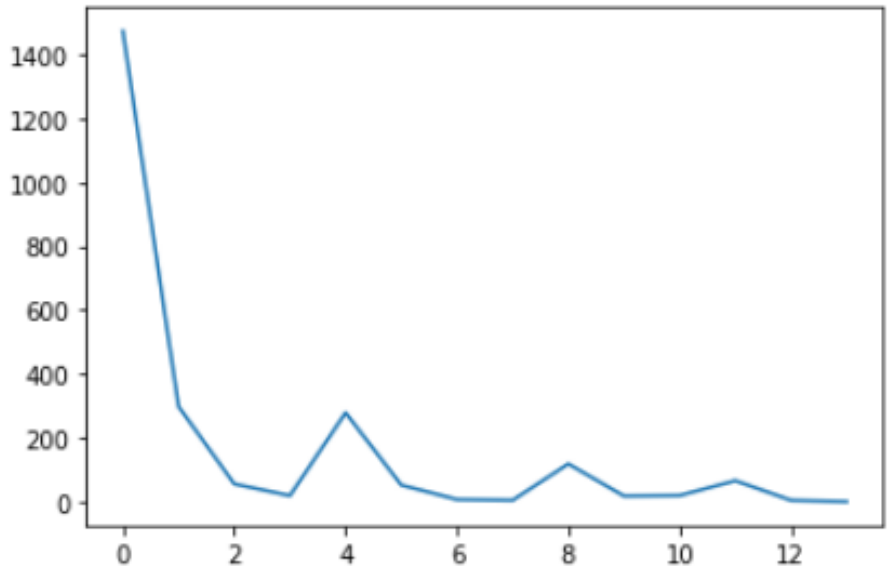
$$Max(I) \rightarrow I, J = (32, 5)$$

Instâncias Mínusculas - Casos especiais

```
plt.plot(zs3 - 165); # 13,12
```

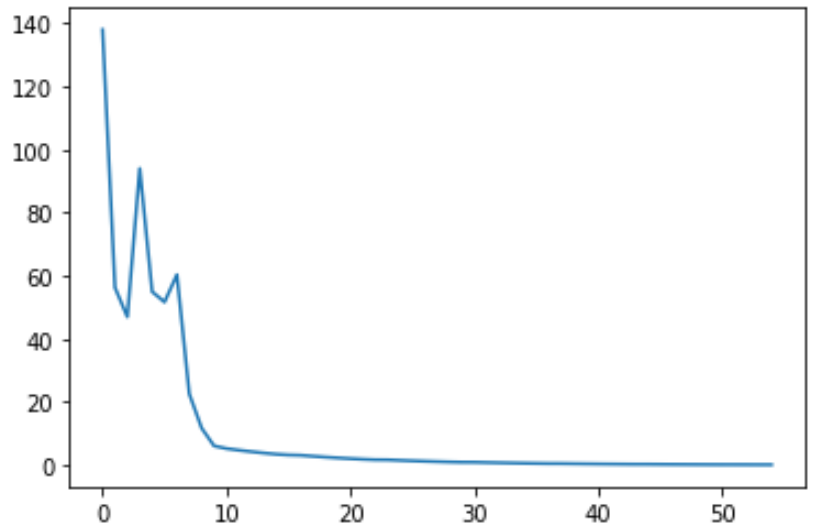



```
plt.plot(zs4 - 239); # 32,5
```

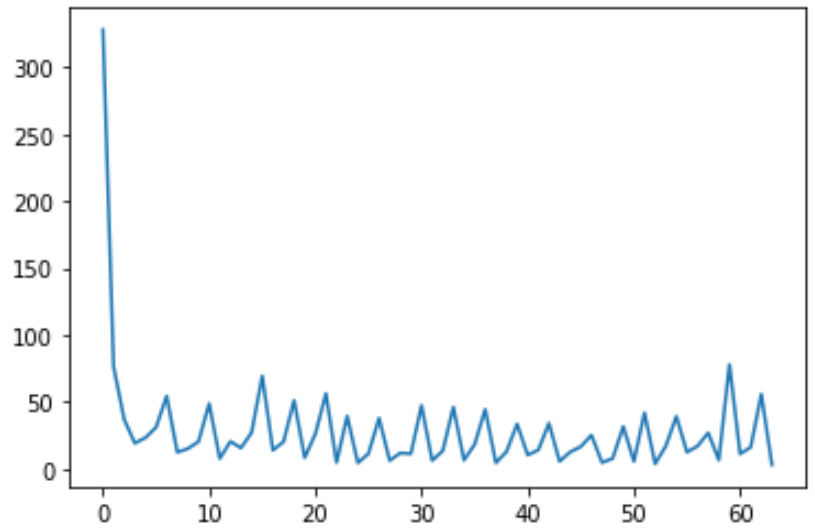


Instancias Maiores

```
plt.plot(zm1 - 98); # 64,6
```

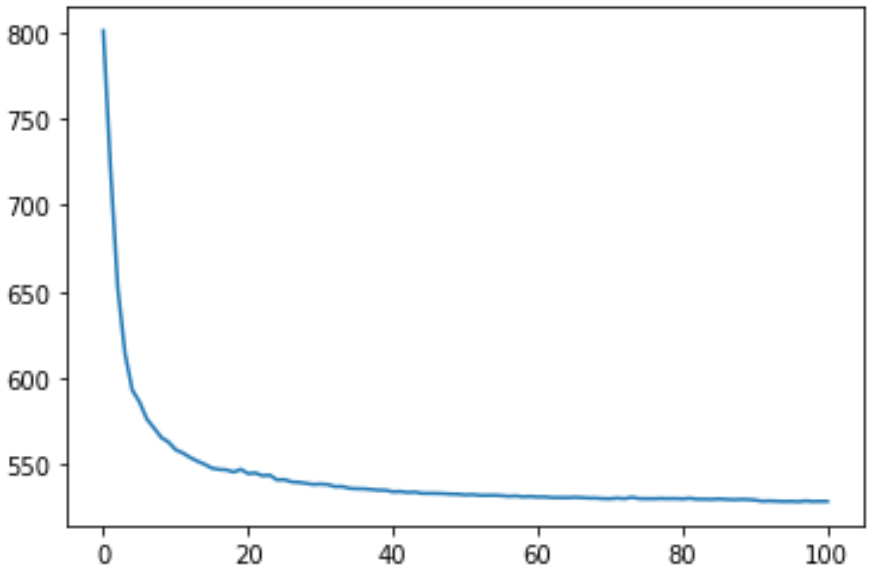



```
plt.plot(zm2 - 61); # 25,25
```

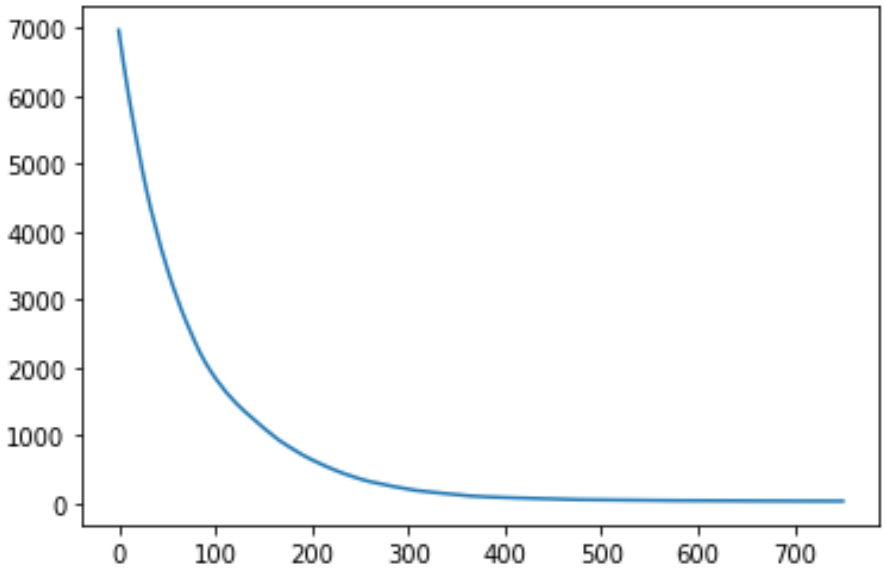


Instancias Maiores Ainda

```
plt.plot(zgig); # 500x100
```




```
plt.plot(zgig2 - 421);   # 200x300
```



Branch and Bound

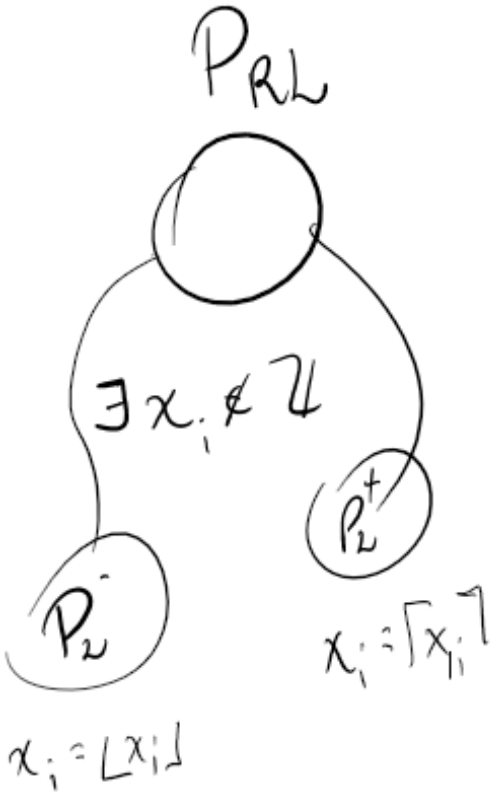
Branch and Bound - preâmbulo

Branch and Bound - preâmbulo

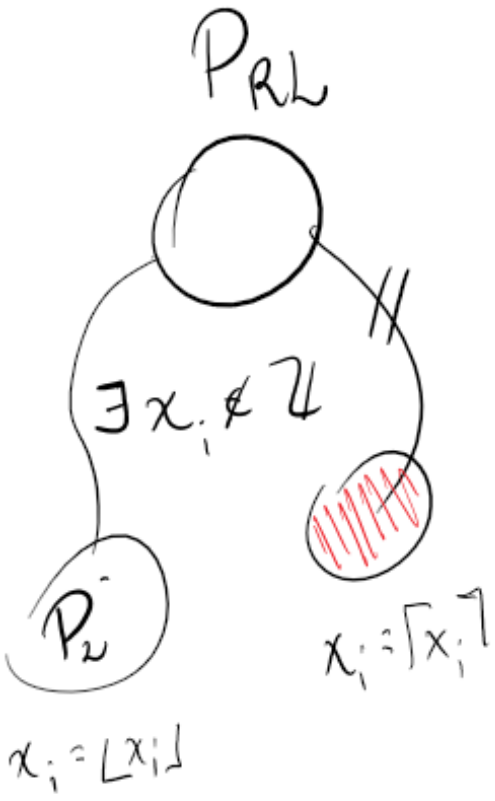
Branch and Bound - preâmbulo

Branch and Bound - preâmbulo

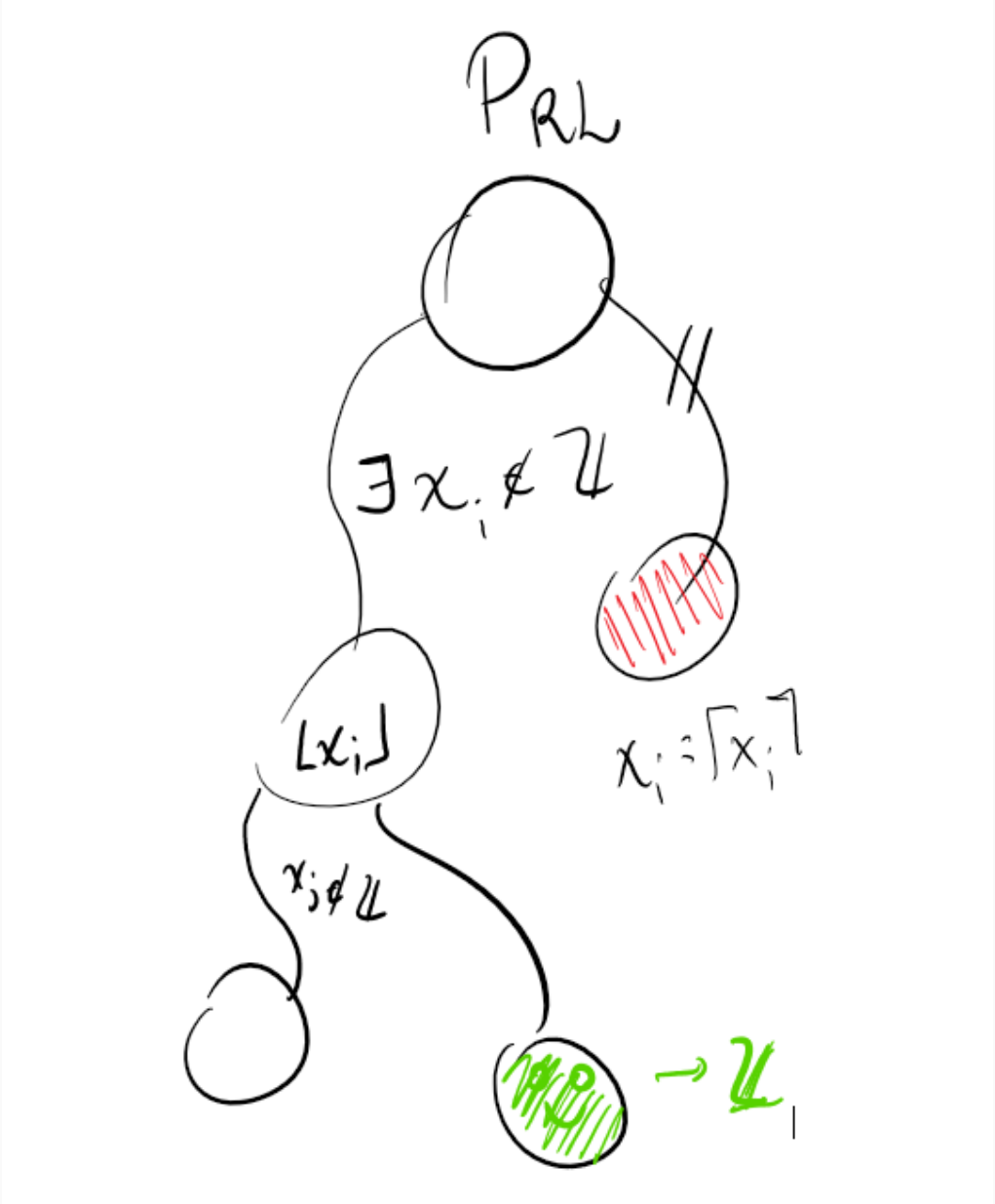
Branch and Bound - preâmbulo



Branch and Bound - preâmbulo



Branch and Bound - preâmbulo



Branch and Bound - Pseudocódigo

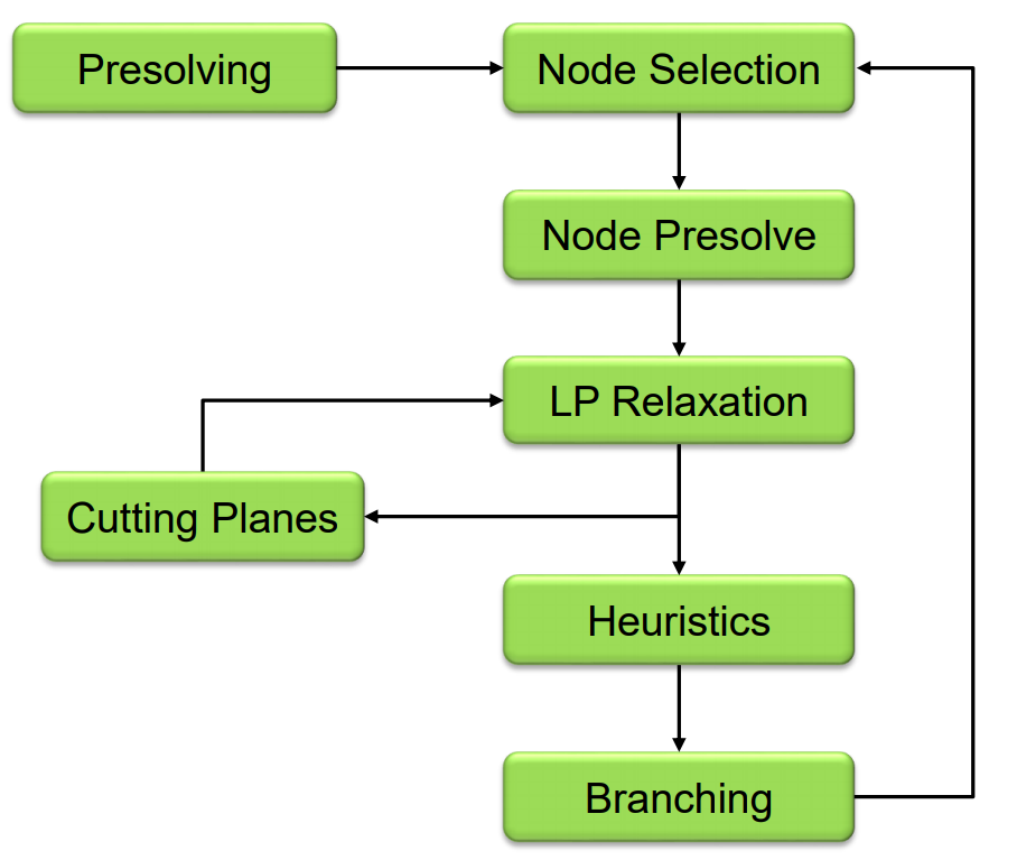
Data: $A_{I\times J}, c \in \mathbb{R}^I_+$
nodes $= [\mathcal{P}_{\mathcal{RL}}]$;
while *nodes* $\neq \varnothing$ **do**
    $p \leftarrow nodes[1]$;
    $p_{sol} =$ **Solve**(p);
    if $p_{sol} \notin \mathbb{Z}$ **then**
        $p_- \leftarrow \lfloor p_{sol} \rfloor$;
        $p_+ \leftarrow \lfloor p_{sol} + 1 \rfloor$;
        nodes $\leftarrow p_+, p_-$;
    end
    if *Infeasible* **then**
        nextnode;
    end
    Temos upperbounds!;
end

Algorithm 2: Rotina do Branch_n_Bound

Branch and Bound

```
Solved in 10 iterations and 0.00 seconds
Optimal objective  1.145000000e+02
----------> Any[1] <--------- BRANCHING

Solved in 11 iterations and 0.00 seconds
Optimal objective  9.633333333e+01
----------> Any[1, 4] <--------- BRANCHING

Solved in 7 iterations and 0.00 seconds
Optimal objective  9.200000000e+01
----------> Any[1, 4, 9] <--------- BRANCHING

--> Any[114.5, 96.33333333333334, 77.0, 92.0] <---
Gurobi Optimizer version 9.0.3 build v9.0.3rc0 (win64)
Optimize a model with 17 rows, 15 columns and 112 nonzeros
Model fingerprint: 0x1216ab33
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [3e+00, 8e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 10 rows and 15 columns
Presolve time: 0.00s

Solved in 0 iterations and 0.00 seconds
Infeasible model
```

Gurobi


Presolving
Node Selection
Node Presolve
LP Relaxation
Cutting Planes
Heuristics
Branching

Branch and Bound - Gurobi

In [1236]:
```
rp, rp_x = setSPP(Gurobi.Optimizer, c, A)

# 500 x 350

println("Solving...");
optimize!(rp)
```

```
Academic license - for non-commercial use only
Solving...
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.3 build v9.0.3rc0 (win64)
Optimize a model with 500 rows, 350 columns and 87330 nonzeros
Model fingerprint: 0xa0d1fe57
Variable types: 0 continuous, 350 integer (350 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 24.0000000
Presolve removed 0 rows and 4 columns
Presolve time: 0.43s
Presolved: 500 rows, 346 columns, 86311 nonzeros
Variable types: 0 continuous, 346 integer (346 binary)

Root relaxation: objective 1.562546e+02, 307 iterations, 0.03 seconds

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

     0     0  156.25463    0   79   24.00000  156.25463   551%     -    0s
H    0     0                      99.0000000  156.25463  57.8%     -    0s
     0     0  124.72377    0   49   99.00000  124.72377  26.0%     -    1s

Explored 1 nodes (1190 simplex iterations) in 1.63 seconds
Thread count was 8 (of 8 available processors)

Solution count 2: 99 24

Optimal solution found (tolerance 1.00e-04)
Best objective 9.900000000000e+01, best bound 9.900000000000e+01, gap 0.0000%
```

Branch and Bound - Gurobi

In [1241]:
```
rp, rp_x = setSPP(Gurobi.Optimizer, c, A)

# 500 x 350

println("Solving...");
set_optimizer_attribute(rp, "Presolve", 0)
set_optimizer_attribute(rp, "Cuts", 0)
set_optimizer_attribute(rp, "Heuristics", 0)
optimize!(rp)
```

```
Academic license - for non-commercial use only
Solving...
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.3 build v9.0.3rc0 (win64)
Optimize a model with 500 rows, 350 columns and 87330 nonzeros
Model fingerprint: 0xa0d1fe57
Variable types: 0 continuous, 350 integer (350 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Variable types: 0 continuous, 350 integer (350 binary)

Root relaxation: objective 1.562546e+02, 307 iterations, 0.03 seconds

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

     0     0  156.25463    0   79          -  156.25463      -     -    0s
     0     0  156.25463    0   79          -  156.25463      -     -    0s
     0     2  156.25463    0   79          -  156.25463      -     -    0s
*    2     2               1      99.0000000  155.00045  56.6%  80.0    1s

Explored 212 nodes (10426 simplex iterations) in 2.13 seconds
Thread count was 8 (of 8 available processors)

Solution count 1: 99

Optimal solution found (tolerance 1.00e-04)
Best objective 9.900000000000e+01, best bound 9.900000000000e+01, gap 0.0000%
```

Branch and Bound - Gurobi

In [1242]:
```
rp, rp_x = setSPP(Gurobi.Optimizer, c, A)

# 500 x 350

println("Solving...");
set_optimizer_attribute(rp, "Presolve", 0)
set_optimizer_attribute(rp, "Cuts", 0)
set_optimizer_attribute(rp, "Heuristics", 0)
set_optimizer_attribute(rp, "CliqueCuts", 1)
optimize!(rp)
```

```
Academic license - for non-commercial use only
Solving...
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.3 build v9.0.3rc0 (win64)
Optimize a model with 500 rows, 350 columns and 87330 nonzeros
Model fingerprint: 0xa0d1fe57
Variable types: 0 continuous, 350 integer (350 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Variable types: 0 continuous, 350 integer (350 binary)

Root relaxation: objective 1.562546e+02, 307 iterations, 0.03 seconds

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

     0     0  156.25463    0   79          -  156.25463      -     -    0s
*    0     0               0      99.0000000   99.00000  0.00%     -    0s

Cutting planes:
  Clique: 1

Explored 1 nodes (443 simplex iterations) in 0.83 seconds
Thread count was 8 (of 8 available processors)

Solution count 1: 99

Optimal solution found (tolerance 1.00e-04)
Best objective 9.900000000000e+01, best bound 9.900000000000e+01, gap 0.0000%
```

Branch and Bound - Gurobi

In [1246]:
```
rp, rp_x = setSPP(Gurobi.Optimizer, c, A)

# 500 x 350

println("Solving...");
set_optimizer_attribute(rp, "Presolve", 0)
set_optimizer_attribute(rp, "Cuts", 0)
set_optimizer_attribute(rp, "Heuristics", 1)
optimize!(rp)
```

```
Academic license - for non-commercial use only
Solving...
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.3 build v9.0.3rc0 (win64)
Optimize a model with 500 rows, 350 columns and 87330 nonzeros
Model fingerprint: 0xa0d1fe57
Variable types: 0 continuous, 350 integer (350 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 24.0000000
Variable types: 0 continuous, 350 integer (350 binary)

Root relaxation: objective 1.562546e+02, 307 iterations, 0.04 seconds

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

     0     0  156.25463    0   79   24.00000  156.25463   551%     -    0s
H    0     0                      99.0000000  156.25463  57.8%     -    0s
     0     0  156.25463    0   79   99.00000  156.25463  57.8%     -    0s
     0     2  156.25463    0   79   99.00000  156.25463  57.8%     -    1s

Explored 213 nodes (10658 simplex iterations) in 4.44 seconds
Thread count was 8 (of 8 available processors)

Solution count 2: 99 24

Optimal solution found (tolerance 1.00e-04)
Best objective 9.900000000000e+01, best bound 9.900000000000e+01, gap 0.0000%
```

Cortes adicionais

In [1254]:
```
rp, rp_x = setSPP(Gurobi.Optimizer, c, A)

# 250 x 1000

println("Solving...");
set_optimizer_attribute(rp, "Presolve", 0)
set_optimizer_attribute(rp, "Cuts", 0)
set_optimizer_attribute(rp, "Heuristics", 0)
set_optimizer_attribute(rp, "CliqueCuts", 1)
optimize!(rp)
```

```
Academic license - for non-commercial use only
Solving...
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.3 build v9.0.3rc0 (win64)
Optimize a model with 1000 rows, 250 columns and 124532 nonzeros
Model fingerprint: 0xc05958ec
Variable types: 0 continuous, 250 integer (250 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Variable types: 0 continuous, 250 integer (250 binary)

Root relaxation: objective 1.406409e+02, 313 iterations, 0.05 seconds

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

     0     0  140.64092    0   68          -  140.64092      -     -    0s
*    0     0               0      98.0000000   98.00000  0.00%     -    0s

Cutting planes:
  Clique: 1

Explored 1 nodes (409 simplex iterations) in 0.94 seconds
Thread count was 8 (of 8 available processors)

Solution count 1: 98

Optimal solution found (tolerance 1.00e-04)
Best objective 9.800000000000e+01, best bound 9.800000000000e+01, gap 0.0000%
```

Cortes adicionais

```
In [1256]: rp, rp_x = setSPP(Gurobi.Optimizer, c, A)

           # 250 x 1000

           println("Solving...");
           set_optimizer_attribute(rp, "Presolve", 0)
           set_optimizer_attribute(rp, "Cuts", 0)
           set_optimizer_attribute(rp, "Heuristics", 0)
           optimize!(rp)
```

```
Academic license - for non-commercial use only
Solving...
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.3 build v9.0.3rc0 (win64)
Optimize a model with 1000 rows, 250 columns and 124532 nonzeros
Model fingerprint: 0xc05958ec
Variable types: 0 continuous, 250 integer (250 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Variable types: 0 continuous, 250 integer (250 binary)

Root relaxation: objective 1.406409e+02, 313 iterations, 0.05 seconds

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

     0     0  140.64092    0   68          -  140.64092      -     -    0s
     0     0  140.64092    0   68          -  140.64092      -     -    0s
     0     2  140.64092    0   68          -  140.64092      -     -    0s
*    2     2               1      78.0000000  140.49189  80.1%  54.0    0s
*    6     2               3      98.0000000  139.67945  42.5%  54.8    0s

Explored 107 nodes (5937 simplex iterations) in 1.97 seconds
Thread count was 8 (of 8 available processors)

Solution count 2: 98 78

Optimal solution found (tolerance 1.00e-04)
Best objective 9.800000000000e+01, best bound 9.800000000000e+01, gap 0.0000%
```

Testes 1000x1000

In [1276]:
```
rp, rp_x = setSPP(Gurobi.Optimizer, c, A)

# 1000 x 1000

println("Solving...");
set_optimizer_attribute(rp, "Presolve", 0)
set_optimizer_attribute(rp, "Cuts", 0)
set_optimizer_attribute(rp, "Heuristics", 1)
optimize!(rp)
```

```
Academic license - for non-commercial use only
Solving...
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.3 build v9.0.3rc0 (win64)
Optimize a model with 1000 rows, 1000 columns and 499993 nonzeros
Model fingerprint: 0x3538eacf
Variable types: 0 continuous, 1000 integer (1000 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 82.0000000
Variable types: 0 continuous, 1000 integer (1000 binary)

Root relaxation: objective 1.672537e+02, 1102 iterations, 0.39 seconds

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

     0     0  167.25367    0  170   82.00000  167.25367   104%     -    1s
H    0     0                       97.0000000  167.25367  72.4%     -    1s
H    0     0                       98.0000000  167.25367  70.7%     -    1s
H    0     0                       99.0000000  167.25367  68.9%     -    1s
     0     0  167.25367    0  170   99.00000  167.25367  68.9%     -    2s
     0     2  166.22779    0  170   99.00000  166.22779  67.9%     -    7s
     5     2  165.95875    3  162   99.00000  165.95875  67.6%   252   10s
    23     2  163.13437   12  154   99.00000  163.52554  65.2%   258   15s
    56     2  159.47683   29  168   99.00000  159.73932  61.4%   238   20s
    77     3  157.58449   39  172   99.00000  157.73620  59.3%   234   25s
   101     2     cutoff   51         99.00000  155.79578  57.4%   226   30s
   153     4  152.48937   78  145   99.00000  152.70973  54.3%   211   35s
   207     2     cutoff  104         99.00000  148.11420  49.6%   203   40s
   267     2  144.06595  135  130   99.00000  144.10628  45.6%   192   46s
   305     4     cutoff  153         99.00000  141.43055  42.9%   187   50s
   354     8  138.39151  178  116   99.00000  138.54903  39.9%   178   55s
   405     2     cutoff  203         99.00000  135.87803  37.3%   170   60s
   453     2     cutoff  227         99.00000  132.77404  34.1%   164   66s
   495     2     cutoff  248         99.00000  129.61580  30.9%   160   70s
   539     2     cutoff  270         99.00000  126.56387  27.8%   155   76s
   579     2     cutoff  290         99.00000  123.96743  25.2%   151   80s
   617     2     cutoff  309         99.00000  121.98558  23.2%   147   86s
   663     2     cutoff  332         99.00000  119.67646  20.9%   143   90s
   691     2     cutoff  346         99.00000  117.95466  19.1%   139   96s
   747     2     cutoff  374         99.00000  112.86426  14.0%   133  101s
   803     2     cutoff  402         99.00000  108.37775  9.47%   126  107s

Explored 857 nodes (103297 simplex iterations) in 107.89 seconds
Thread count was 8 (of 8 available processors)

Solution count 4: 99 98 97 82
```

Testes 1000x1000

In [1274]:
```
rp, rp_x = setSPP(Gurobi.Optimizer, c, A)

# 1000 x 1000

println("Solving...");
set_optimizer_attribute(rp, "Presolve", 0)
set_optimizer_attribute(rp, "Cuts", 0)
set_optimizer_attribute(rp, "Heuristics", 0)
set_optimizer_attribute(rp, "CliqueCuts", 1)
optimize!(rp)
```

```
Academic license - for non-commercial use only
Solving...
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.3 build v9.0.3rc0 (win64)
Optimize a model with 1000 rows, 1000 columns and 499993 nonzeros
Model fingerprint: 0x3538eacf
Variable types: 0 continuous, 1000 integer (1000 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Variable types: 0 continuous, 1000 integer (1000 binary)

Root relaxation: objective 1.672537e+02, 1102 iterations, 0.38 seconds

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

     0     0  167.25367    0  170          -  167.25367      -     -    1s
     0     0  167.25367    0  170          -  167.25367      -     -    2s
     0     2  167.25367    0  170          -  167.25367      -     -    3s
*    2     2               1      92.0000000  167.22688  81.8%   252    4s
     5     2  166.71281    3  170   92.00000  166.71281  81.2%   243    5s
*    6     2               3      97.0000000  166.70912  71.9%   245    5s
*   12     2               6      98.0000000  165.74325  69.1%   244    8s
    19     2  164.16287   10  162   98.00000  164.55984  67.9%   258   10s
*   20     2              10      99.0000000  164.15799  65.8%   259   10s
    81     3     cutoff   41        99.00000  158.21389  59.8%   242   15s
   191     5     cutoff   96        99.00000  150.51555  52.0%   214   20s
   328     5     cutoff  164        99.00000  140.64486  42.1%   190   25s
   472     2  131.72729  236  124   99.00000  131.72729  33.1%   175   30s
   625     1     cutoff  313        99.00000  120.94933  22.2%   157   35s

Explored 831 nodes (110759 simplex iterations) in 37.61 seconds
Thread count was 8 (of 8 available processors)

Solution count 4: 99 98 97 92

Optimal solution found (tolerance 1.00e-04)
Best objective 9.900000000000e+01, best bound 9.900000000000e+01, gap 0.0000%
```

Testes 1000x1000

In [1272]:
```
rp, rp_x = setSPP(Gurobi.Optimizer, c, A)

# 1000 x 1000

println("Solving...");
optimize!(rp)
```

```
Academic license - for non-commercial use only
Solving...
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.3 build v9.0.3rc0 (win64)
Optimize a model with 1000 rows, 1000 columns and 499993 nonzeros
Model fingerprint: 0x3538eacf
Variable types: 0 continuous, 1000 integer (1000 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 82.0000000
Presolve removed 0 rows and 6 columns
Presolve time: 4.02s
Presolved: 1000 rows, 994 columns, 497022 nonzeros
Variable types: 0 continuous, 994 integer (994 binary)

Root relaxation: objective 1.672537e+02, 1102 iterations, 0.45 seconds

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

     0     0  165.73086    0  128   82.00000  165.73086   102%     -    9s
H    0     0                      99.0000000  165.73086  67.4%     -    9s
     0     0  140.35736    0  123   99.00000  140.35736  41.8%     -   11s

Explored 1 nodes (6181 simplex iterations) in 12.25 seconds
Thread count was 8 (of 8 available processors)

Solution count 2: 99 82

Optimal solution found (tolerance 1.00e-04)
Best objective 9.900000000000e+01, best bound 9.900000000000e+01, gap 0.0000%
```

Outras heurísticas?

- Algoritmos genéticos! (Muito lento)
- Delorme, Gandibleux, and Rodriguez, “GRASP for set packing problems”

Referências

References

- Delorme, Xavier, Xavier Gandibleux, and Joaquin Rodriguez. "GRASP for set packing problems". In: *European Journal of Operational Research* 153.3 (2004), pp. 564–580.
- Guo, Yunsong et al. "Using a Lagrangian heuristic for a combinatorial auction problem". In: *17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)*. IEEE. 2005, 5–pp.
- Gurobi Optimization, Incorporate. "Gurobi optimizer reference manual". In: *URL http://www. gurobi. com* (2018).
- Karp, Richard M. "Reducibility among combinatorial problems". In: *Complexity of computer computations*. Springer, 1972, pp. 85–103.

 Lusby, Richard et al. “Routing trains through railway junctions: a new set-packing approach”. In: *Transportation Science* 45.2 (2011), pp. 228–245.

Agradecimentos

Obrigado!