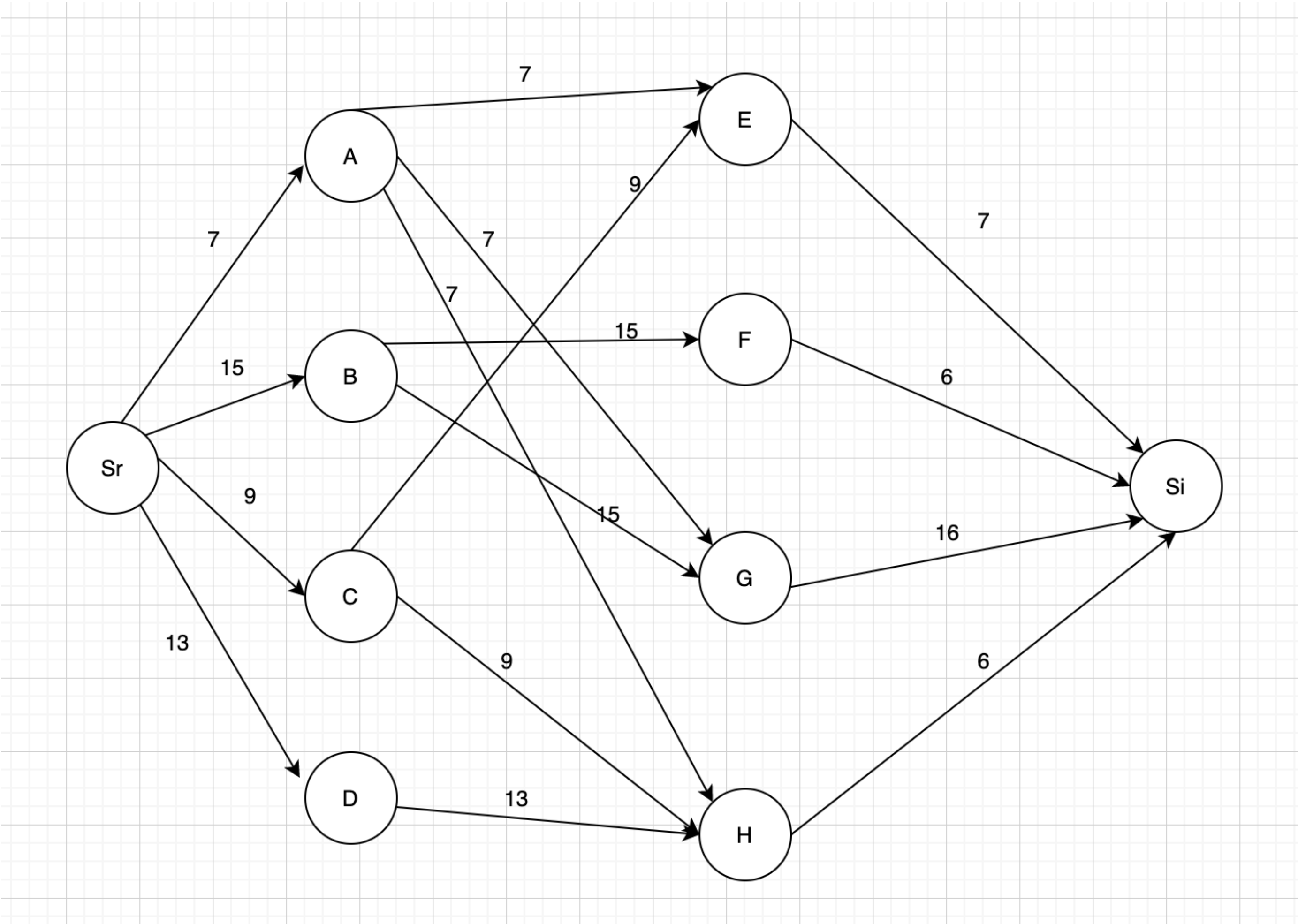
Problem 1



(b)

In [37]:

```
using JuMP, Clp
m = Model(Clp.Optimizer)

# create list of nodes. create a dummy source node and dummy sink node.
nodes = [:sr,:A,:B,:C,:D,:E,:F,:G,:H,:si]

# create list of all arcs in the network.
arcs = [(:sr,:A), (:sr,:B), (:sr,:C), (:sr,:D), (:A,:E), (:A,:G), (:A,:H), (:B,:F),
    (:B,:G), (:C,:E), (:C,:H), (:D,:H), (:E,:si), (:F,:si), (:G,:si), (:H,:si), (:si,:sr)]

# dictionary of arc capacities, making dummy cap "big" enough
capacity = Dict(zip(arcs,[7 15 9 13 7 7 7 15 15 9 9 13 100 100 100 100 100]))


#variables represent flow on each arc
@variable(m, x[arcs] >= 0)
# maximize total flow on arc from sink to source
@objective(m, Min, -x[(:si,:sr)])

@constraint(m, cap[a in arcs], x[a] <= capacity[a]) # obey capacity restrictions # balance flow on 
@constraint(m, flow[i in nodes], sum(x[a] for a in arcs if a[1] == i) == sum(x[a] for a in arcs i

set_optimizer_attribute(m, "LogLevel", 0)
#We aren't meeting the demand, so add a constraint and re-solve:
optimize!(m)
println("Total flow through network: ", -objective_value(m))
println("Flow on each arc: ", value.(x))
println("check if demand met: ")
println("flow to E (should be at least 7): ", value(x[(:E,:si)]))
println("flow to F (should be at least 6): ", value(x[(:F,:si)]))
println("flow to G (should be at least 16): ", value(x[(:G,:si)]))
println("flow to H (should be at least 6): ", value(x[(:H,:si)]))
```

```
Total flow through network: 44.0
Flow on each arc: 1-dimensional DenseAxisArray{Float64,1,...} with index sets:
    Dimension 1, Tuple{Symbol,Symbol}[(:sr, :A), (:sr, :B), (:sr, :C), (:sr, :D), (:
A, :E), (:A, :G), (:A, :H), (:B, :F), (:B, :G), (:C, :E), (:C, :H), (:D, :H), (:E, :
si), (:F, :si), (:G, :si), (:H, :si), (:si, :sr)]
And data, a 17-element Array{Float64,1}:
  7.0
 15.0
  9.0
 13.0
  0.0
  0.0
  7.0
 15.0
  0.0
  9.0
  0.0
 13.0
  9.0
 15.0
  0.0
 20.0
 44.0
check if demand met:
```

```
flow to E (should be at least 7): 9.0
flow to F (should be at least 6): 15.0
flow to G (should be at least 16): 0.0
flow to H (should be at least 6): 20.0
```

In [38]:

```
@constraint(m, x[(:G,:si)]==16)

optimize!(m)
println("Total flow through network: ", -objective_value(m))
println("Flow on each arc: ", value.(x))
println("check if demand met: ")
println("flow to E (should be at least 7): ", value(x[(:E,:si)]))
println("flow to F (should be at least 6): ", value(x[(:F,:si)]))
println("flow to G (should be at least 16): ", value(x[(:G,:si)]))
println("flow to H (should be at least 6): ", value(x[(:H,:si)]))
```

```
Total flow through network: 44.0
Flow on each arc: 1-dimensional DenseAxisArray{Float64,1,...} with index sets:
    Dimension 1, Tuple{Symbol,Symbol}[(:sr, :A), (:sr, :B), (:sr, :C), (:sr, :D), (:
A, :E), (:A, :G), (:A, :H), (:B, :F), (:B, :G), (:C, :E), (:C, :H), (:D, :H), (:E, :
si), (:F, :si), (:G, :si), (:H, :si), (:si, :sr)]
And data, a 17-element Array{Float64,1}:
  7.0
 15.0
  9.0
 13.0
  6.0
  1.0
  0.0
  0.0
 15.0
  9.0
  0.0
 13.0
 15.0
  0.0
 16.0
 13.0
 44.0
check if demand met:
flow to E (should be at least 7): 15.0
flow to F (should be at least 6): 0.0
flow to G (should be at least 16): 16.0
flow to H (should be at least 6): 13.0
```

In [39]:

```
@constraint(m, x[(:F,:si)]==6)

optimize!(m)
println("Total flow through network: ", -objective_value(m))
println("Flow on each arc: ", value.(x))
println("check if demand met: ")
println("flow to E (should be at least 7): ", value(x[(:E,:si)]))
println("flow to F (should be at least 6): ", value(x[(:F,:si)]))
println("flow to G (should be at least 16): ", value(x[(:G,:si)]))
println("flow to H (should be at least 6): ", value(x[(:H,:si)]))
```

```
Total flow through network: 44.0
Flow on each arc: 1-dimensional DenseAxisArray{Float64,1,...} with index sets:
    Dimension 1, Tuple{Symbol,Symbol}[(:sr, :A), (:sr, :B), (:sr, :C), (:sr, :D), (:
A, :E), (:A, :G), (:A, :H), (:B, :F), (:B, :G), (:C, :E), (:C, :H), (:D, :H), (:E, :
si), (:F, :si), (:G, :si), (:H, :si), (:si, :sr)]
And data, a 17-element Array{Float64,1}:
  7.0
 15.0
  9.0
 13.0
  0.0
  7.0
  0.0
  6.0
  9.0
  9.0
  0.0
 13.0
  9.0
  6.0
 16.0
 13.0
 44.0
check if demand met:
flow to E (should be at least 7): 9.0
flow to F (should be at least 6): 6.0
flow to G (should be at least 16): 16.0
flow to H (should be at least 6): 13.0
```

(c)

This model (max flow with two extra constraints) gives a feasible flow that meets demand!

Now we need to find a minimum cut. Remember that max flow = min cut, so we need to find a setof arcs that separate the source from the sink such that the sum of the capacities on the arcs = 44.We can use dual varaiable values to recover a minimum cut

In [40]:

```
min_cut=0
for a in arcs# if the dual variable is nonzero, the primal capacity constraint is active
    if abs(dual(cap[a])) > 10e-5
        # print the arc where the associated primal capacity is active
        println("Arcincut:", a,"(Capacity:", capacity[a],")")
        min_cut=min_cut+capacity[a]
    end
end
println("Total capacity of this minimum cut(should be 44):", min_cut)
```

```
Arcincut:(:sr, :A)(Capacity:7)
Arcincut:(:sr, :B)(Capacity:15)
Arcincut:(:sr, :C)(Capacity:9)
Arcincut:(:sr, :D)(Capacity:13)
Total capacity of this minimum cut(should be 44):44
```

44 is the optimal solution according to the Complementary Slackness Theorem because all are active constraints(from the sr to ABCD).

Problem 2

(a)

primal linear program:

In [10]:

```
using JuMP, Clp
m = Model(Clp.Optimizer)

@variable(m, x1 >= 0)
@variable(m, x2 >= 0)
@variable(m, x3 >= 0)

@constraint(m, con1,   x1 + 2*x2 + 2*x3 <= 3)
@constraint(m, con2,  2*x1 - x2 + 3*x3 == 3)

@objective(m, Max, 2*x1 + x2 + 4*x3)           # maximize p

# solve this instance of the Top Brass problem
optimize!(m)

# print out the full model and solution
display(m)

println("x1 = ", value(x1))
println("x2 = ", value(x2))
println("x3 = ", value(x3))

println("p_max will be \$", objective_value(m))
```

$$\begin{aligned} \max \quad & 2x1 + x2 + 4x3 \\ \text{Subject to} \quad & 2x1 - x2 + 3x3 = 3.0 \\ & x1 + 2x2 + 2x3 \leq 3.0 \\ & x1 \geq 0.0 \\ & x2 \geq 0.0 \\ & x3 \geq 0.0 \end{aligned}$$

```
x1 = 0.0
x2 = 0.375
x3 = 1.125
p_max will be $4.875
Coin0506I Presolve 2 (0) rows, 3 (0) columns and 6 (0) elements
Clp0006I 0  Obj 0 Primal inf 0.9999999 (1) Dual inf 7.9999997 (3)
Clp0006I 3  Obj 4.875
Clp0000I Optimal - objective value 4.875
Clp0032I Optimal objective 4.875 - 3 iterations time 0.002
```

Dual linear program

In [11]:

```
using JuMP, Clp
m = Model(Clp.Optimizer)
@variable(m, λ[1:2] >= 0) # variables for each primal constraint

# constraints ensuring the correct relationship with each primal variable
# to guarantee an upper bound
@constraint(m, 2λ[1] + λ[2] >= 2)
@constraint(m, 2λ[2] - λ[1] >= 1)
@constraint(m, 2λ[2] + 3λ[1] >= 4)

# objective is to minimize the upper bound on the primal solution
@objective(m, Min, 3*λ[2] + 3*λ[1] )

# solve this instance of the Top Brass dual
optimize!(m)

# print the dual model and solution
display(m)

println("dual variables are: ", value.(λ))
println("Optimal objective is: ", objective_value(m))
```

$$
\begin{aligned}
\min \quad & 3\lambda_2 + 3\lambda_1 \\
\text{Subject to} \quad & 2\lambda_1 + \lambda_2 \geq 2.0 \\
& 2\lambda_2 - \lambda_1 \geq 1.0 \\
& 2\lambda_2 + 3\lambda_1 \geq 4.0 \\
& \lambda_1 \geq 0.0 \\
& \lambda_2 \geq 0.0
\end{aligned}
$$

```
dual variables are: [0.75, 0.875]
Optimal objective is: 4.875
Coin0506I Presolve 3 (0) rows, 2 (0) columns and 6 (0) elements
Clp0006I 0  Obj 0 Primal inf 2.833333 (3)
Clp0006I 2  Obj 4.875
Clp0000I Optimal - objective value 4.875
Clp0032I Optimal objective 4.875 - 2 iterations time 0.002
```

using the weak duality theorem, it is clear that $p \leq p* \leq d* \leq d$. (where p is the objective function from Primal problem, d is the objective function from Dual problem.) Since $d* = 4.875$, it follows that $p* \leq 4.875 < 6$.

(b)

primal linear program:

In [48]:

```
using JuMP, Cbc
m = Model(Cbc.Optimizer)

@variable(m, y1 >= 0)      #y1 = -x1
@variable(m, y2)
@variable(m, x3 >= 0)
@variable(m, x4 >= 0)


@constraint(m, con1,   4*y2 + 6*x3 <= 36)
@constraint(m, con2,   x3 + x4 <= 16 )
@constraint(m, con3,   -y1 + y2 == 4)


@objective(m, Max, -6*y1 + 2*(y2) + 4*x3 + x4)              # maximize p

# solve this instance of the Top Brass problem
optimize!(m)

# print out the full model and solution
display(m)

println("x1 = ", value(-y1))
println("x2 = ", value(-y2))
println("x3 = ", value(x3))
println("x4 = ", value(x4))

println("p_max will be \$", objective_value(m))
```

$$
\begin{aligned}
\max\quad & -6y1 + 2y2 + 4x3 + x4\\
\text{Subject to}\quad & -y1 + y2 = 4.0\\
& 4y2 + 6x3 \leq 36.0\\
& x3 + x4 \leq 16.0\\
& y1 \geq 0.0\\
& x3 \geq 0.0\\
& x4 \geq 0.0
\end{aligned}
$$

```
x1 = 0.0
x2 = -4.0
x3 = 3.3333333333333335
x4 = 12.666666666666666
p_max will be $34.0
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: May 23 2020

command line - Cbc_C_Interface -solve -quit (default strategy 1)
Presolve 0 (-3) rows, 0 (-4) columns and 0 (-6) elements
```

```
Empty problem - 0 rows, 0 columns and 0 elements
Optimal - objective value 34
After Postsolve, objective 34, infeasibilities - dual 0 (0), primal 0 (0)
Optimal objective 34 - 0 iterations time 0.002, Presolve 0.00
Total time (CPU seconds):       0.00   (Wallclock seconds):       0.00
```

Dual linear program:

In [50]:

```
using JuMP, Clp
m = Model(Clp.Optimizer)
@variable(m, λ[1:3] >= 0) # variables for each primal constraint

# constraints ensuring the correct relationship with each primal variable
# to guarantee an upper bound
@constraint(m, -λ[3] >= -6)
@constraint(m, λ[3] + 4*λ[1] >= 2)
@constraint(m, 6λ[1] + λ[2] >= 4)
@constraint(m, λ[2] >= 1)

# objective is to minimize the upper bound on the primal solution
@objective(m, Min, 4*λ[3] + 36*λ[1] + 16*λ[2])

# solve this instance of the Top Brass dual
optimize!(m)

# print the dual model and solution
display(m)

println("dual variables are: ", value.(λ))
println("Optimal objective is: ", objective_value(m) + 8)
```

$$
\begin{aligned}
\min\quad & 4\lambda_3 + 36\lambda_1 + 16\lambda_2\\
\text{Subject to}\quad & -\lambda_3 \geq -6.0\\
& \lambda_3 + 4\lambda_1 \geq 2.0\\
& 6\lambda_1 + \lambda_2 \geq 4.0\\
& \lambda_2 \geq 1.0\\
& \lambda_1 \geq 0.0\\
& \lambda_2 \geq 0.0\\
& \lambda_3 \geq 0.0
\end{aligned}
$$

```
dual variables are: [0.5, 1.0, 0.0]
Optimal objective is: 42.0
Coin0506I Presolve 2 (-2) rows, 3 (0) columns and 4 (-2) elements
Clp0006I 0  Obj 16 Primal inf 0.999998 (2)
Clp0006I 1  Obj 34
Clp0000I Optimal - objective value 34
Coin0511I After Postsolve, objective 34, infeasibilities - dual 0 (0), primal 0 (0)
Clp0032I Optimal objective 34 - 1 iterations time 0.002, Presolve 0.00
```

pf: The solution $x = (0, -4, \frac{10}{3}, \frac{38}{3})$ is feasible solution. The primal constraints x1, x2, x3, x4 do not have slacks. So it is optimal solution to primal problem.

$\lambda = (\, 0, \frac{1}{2}, -1)$

Problem 3

(a)

primal solutions

In [4]:

```julia
using JuMP, Cbc
m = Model(Cbc.Optimizer)

@variable(m, x1 >= 0, Int)
@variable(m, x2 >= 0, Int)
@variable(m, x3 >= 0, Int)
@variable(m, x4 >= 0, Int)


@constraint(m, con1,   2*x1+ 3*x2+ 3*x3+ 5*x4 <= 12000)
@constraint(m, con2,  5*x1+ 5*x2+ 10*x3+ 15*x4 <= 32000)
@constraint(m, con3,   0.25*x1+ x2+ 2*x3+ 3.5*x4 <= 5000 )


@objective(m, Max, 60*x1+ 120*x2+ 200*x3+ 300*x4)              # maximize profit

# solve this instance of the Top Brass problem
optimize!(m)

# print out the full model and solution
display(m)

println("x1 = ", value(x1))
println("x2 = ", value(x2))
println("x3 = ", value(x3))
println("x4 = ", value(x4))

println("max profit will be \$", objective_value(m))
```

$$
\begin{aligned}
\max \quad & 60x1 + 120x2 + 200x3 + 300x4 \\
\text{Subject to} \quad & 2x1 + 3x2 + 3x3 + 5x4 \leq 12000.0 \\
& 5x1 + 5x2 + 10x3 + 15x4 \leq 32000.0 \\
& 0.25x1 + x2 + 2x3 + 3.5x4 \leq 5000.0 \\
& x1 \geq 0.0 \\
& x2 \geq 0.0 \\
& x3 \geq 0.0 \\
& x4 \geq 0.0 \\
& x1 integer \\
& x2 integer \\
& x3 integer \\
& x4 integer
\end{aligned}
$$

```
x1 = 1866.9999999999998
x2 = 977.0
x3 = 1778.0
x4 = 0.0
max profit will be $584860.0
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: May 23 2020

command line - Cbc_C_Interface -solve -quit (default strategy 1)
Continuous objective value is 584889 - 0.00 seconds
Cgl0004I processed model has 3 rows, 4 columns (4 integer (0 of which binary)) an
d 12 elements
```

```
Cbc0012I Integer solution of -584600 found by DiveCoefficient after 0 iterations
and 0 nodes (0.00 seconds)
Cbc0038I Full problem 3 rows 4 columns, reduced to 3 rows 3 columns
Cbc0012I Integer solution of -584800 found by DiveCoefficient after 13 iterations
and 0 nodes (0.03 seconds)
Cbc0031I 3 added rows had average density of 4
Cbc0013I At root node, 3 cuts changed objective from -584888.89 to -584860 in 8 p
asses
Cbc0014I Cut generator 0 (Probing) - 0 row cuts average 0.0 elements, 0 column cu
ts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 1 (Gomory) - 16 row cuts average 4.0 elements, 0 column cu
ts (0 active)  in 0.001 seconds - new frequency is 1
Cbc0014I Cut generator 2 (Knapsack) - 0 row cuts average 0.0 elements, 0 column c
uts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 3 (Clique) - 0 row cuts average 0.0 elements, 0 column cut
s (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 4 (MixedIntegerRounding2) - 0 row cuts average 0.0 element
s, 0 column cuts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 5 (FlowCover) - 0 row cuts average 0.0 elements, 0 column
cuts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0016I Integer solution of -584860 found by strong branching after 13 iteration
s and 0 nodes (0.03 seconds)
Cbc0001I Search completed - best objective -584860, took 13 iterations and 0 node
s (0.03 seconds)
Cbc0032I Strong branching done 4 times (5 iterations), fathomed 1 nodes and fixed
0 variables
Cbc0035I Maximum depth 0, 0 variables fixed on reduced cost
Cuts at root node changed objective from -584889 to -584860
Probing was tried 8 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.000 seconds)
Gomory was tried 8 times and created 16 cuts of which 0 were active after adding
rounds of cuts (0.001 seconds)
Knapsack was tried 8 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.000 seconds)
Clique was tried 8 times and created 0 cuts of which 0 were active after adding r
ounds of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 8 times and created 0 cuts of which 0 were active
after adding rounds of cuts (0.000 seconds)
FlowCover was tried 8 times and created 0 cuts of which 0 were active after addin
g rounds of cuts (0.000 seconds)
TwoMirCuts was tried 1 times and created 0 cuts of which 0 were active after addi
ng rounds of cuts (0.000 seconds)
ZeroHalf was tried 1 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.019 seconds)

Result - Optimal solution found

Objective value:                584860.00000000
Enumerated nodes:               0
Total iterations:               13
Time (CPU seconds):             0.03
Time (Wallclock seconds):       0.02

Total time (CPU seconds):       0.03   (Wallclock seconds):       0.02
```

dual solutions

In [5]:

```
using JuMP, Cbc
m = Model(Cbc.Optimizer)
@variable(m, λ[1:3] >= 0, Int) # variables for each primal constraint

# constraints ensuring the correct relationship with each primal variable
# to guarantee an upper bound
@constraint(m, 2λ[1] + 5λ[2] + 0.25λ[3] >= 60)
@constraint(m, 3λ[1] + 5λ[2] + λ[3] >= 120)
@constraint(m, 3λ[1] + 10λ[2] + 2λ[3] >= 200)
@constraint(m, 5λ[1] + 15λ[2] + 3.5λ[3] >= 300)

# objective is to minimize the upper bound on the primal solution
@objective(m, Min, 12000λ[1] + 32000λ[2] + 5000λ[3] )

# solve this instance of the Top Brass dual
optimize!(m)

# print the dual model and solution
display(m)

println("dual variables are: ", value.(λ))
println("Optimal objective is: ", objective_value(m))
```

$$
\begin{aligned}
\min \quad & 12000\lambda_1 + 32000\lambda_2 + 5000\lambda_3 \\
\text{Subject to} \quad & 2\lambda_1 + 5\lambda_2 + 0.25\lambda_3 \geq 60.0 \\
& 3\lambda_1 + 5\lambda_2 + \lambda_3 \geq 120.0 \\
& 3\lambda_1 + 10\lambda_2 + 2\lambda_3 \geq 200.0 \\
& 5\lambda_1 + 15\lambda_2 + 3.5\lambda_3 \geq 300.0 \\
& \lambda_1 \geq 0.0 \\
& \lambda_2 \geq 0.0 \\
& \lambda_3 \geq 0.0 \\
& \lambda_1 \; integer \\
& \lambda_2 \; integer \\
& \lambda_3 \; integer
\end{aligned}
$$

```
dual variables are: [13.0, 4.0, 61.0]
Optimal objective is: 589000.0
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: May 23 2020

command line - Cbc_C_Interface -solve -quit (default strategy 1)
Continuous objective value is 584889 - 0.00 seconds
Cgl0003I 0 fixed, 3 tightened bounds, 0 strengthened rows, 0 substitutions
Cgl0004I processed model has 4 rows, 3 columns (3 integer (0 of which binary)) and 1
2 elements
Cbc0012I Integer solution of 611000 found by DiveCoefficient after 0 iterations and
0 nodes (0.00 seconds)
Cbc0012I Integer solution of 594000 found by DiveCoefficient after 171 iterations an
d 0 nodes (0.06 seconds)
Cbc0031I 3 added rows had average density of 3
Cbc0013I At root node, 3 cuts changed objective from 584888.89 to 588809.43 in 100 p
asses
```

```
Cbc0014I Cut generator 0 (Probing) - 0 row cuts average 0.0 elements, 1 column cuts
(1 active)  in 0.004 seconds - new frequency is -100
Cbc0014I Cut generator 1 (Gomory) - 145 row cuts average 3.0 elements, 0 column cuts
(0 active)  in 0.007 seconds - new frequency is 1
Cbc0014I Cut generator 2 (Knapsack) - 0 row cuts average 0.0 elements, 0 column cuts
(0 active)  in 0.002 seconds - new frequency is -100
Cbc0014I Cut generator 3 (Clique) - 0 row cuts average 0.0 elements, 0 column cuts
(0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 4 (MixedIntegerRounding2) - 1 row cuts average 3.0 elements,
0 column cuts (0 active)  in 0.004 seconds - new frequency is -100
Cbc0014I Cut generator 5 (FlowCover) - 0 row cuts average 0.0 elements, 0 column cut
s (0 active)  in 0.004 seconds - new frequency is -100
Cbc0010I After 0 nodes, 1 on tree, 594000 best solution, best possible 588809.43 (0.
06 seconds)
Cbc0016I Integer solution of 593000 found by strong branching after 189 iterations a
nd 1 nodes (0.06 seconds)
Cbc0012I Integer solution of 591000 found by DiveCoefficient after 196 iterations an
d 3 nodes (0.06 seconds)
Cbc0016I Integer solution of 589000 found by strong branching after 200 iterations a
nd 4 nodes (0.06 seconds)
Cbc0001I Search completed - best objective 589000, took 200 iterations and 4 nodes
(0.06 seconds)
Cbc0032I Strong branching done 14 times (23 iterations), fathomed 1 nodes and fixed
1 variables
Cbc0035I Maximum depth 1, 0 variables fixed on reduced cost
Cuts at root node changed objective from 584889 to 588809
Probing was tried 100 times and created 1 cuts of which 0 were active after adding r
ounds of cuts (0.004 seconds)
Gomory was tried 115 times and created 164 cuts of which 0 were active after adding
rounds of cuts (0.008 seconds)
Knapsack was tried 100 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.002 seconds)
Clique was tried 100 times and created 0 cuts of which 0 were active after adding ro
unds of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 100 times and created 1 cuts of which 0 were active
after adding rounds of cuts (0.004 seconds)
FlowCover was tried 100 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.004 seconds)
TwoMirCuts was tried 1 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.000 seconds)
ZeroHalf was tried 1 times and created 0 cuts of which 0 were active after adding ro
unds of cuts (0.009 seconds)

Result - Optimal solution found

Objective value:                589000.00000000
Enumerated nodes:               4
Total iterations:               200
Time (CPU seconds):             0.07
Time (Wallclock seconds):       0.05

Total time (CPU seconds):       0.07   (Wallclock seconds):       0.05
```

(b)

In [32]:

```
using JuMP, Cbc
m = Model(Cbc.Optimizer)

@variable(m, x1 >= 0, Int)
@variable(m, x2 >= 0, Int)
@variable(m, x3 >= 0, Int)
@variable(m, x4 >= 0, Int)


@constraint(m, con1,   2*x1+ 3*x2+ 3*x3+ 5*x4 <= 12001)
@constraint(m, con2,  5*x1+ 5*x2+ 10*x3+ 15*x4 <= 32000)
@constraint(m, con3,   0.25*x1+ x2+ 2*x3+ 3.5*x4 <= 5000 )


@objective(m, Max, 60*x1+ 120*x2+ 200*x3+ 300*x4)          # maximize profit

# solve this instance of the Top Brass problem
optimize!(m)

# print out the full model and solution
display(m)

println("x1 = ", value(x1))
println("x2 = ", value(x2))
println("x3 = ", value(x3))
println("x4 = ", value(x4))

println("max profit will be \$", objective_value(m))
```

$$
\begin{aligned}
\max\quad & 60x1 + 120x2 + 200x3 + 300x4\\
\text{Subject to}\quad & 2x1 + 3x2 + 3x3 + 5x4 \leq 12001.0\\
& 5x1 + 5x2 + 10x3 + 15x4 \leq 32000.0\\
& 0.25x1 + x2 + 2x3 + 3.5x4 \leq 5000.0\\
& x1 \geq 0.0\\
& x2 \geq 0.0\\
& x3 \geq 0.0\\
& x4 \geq 0.0\\
& x1 integer\\
& x2 integer\\
& x3 integer\\
& x4 integer
\end{aligned}
$$

```
x1 = 1866.9999999999998
x2 = 977.0
x3 = 1778.0
x4 = 0.0
max profit will be $584860.0
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: May 23 2020

command line - Cbc_C_Interface -solve -quit (default strategy 1)
```

```
Continuous objective value is 584902 - 0.00 seconds
Cgl0004I processed model has 3 rows, 4 columns (4 integer (0 of which binary)) an
d 12 elements
Cbc0012I Integer solution of -584720 found by DiveCoefficient after 0 iterations
and 0 nodes (0.00 seconds)
Cbc0038I Full problem 3 rows 4 columns, reduced to 3 rows 3 columns
Cbc0012I Integer solution of -584780 found by DiveCoefficient after 92 iterations
and 0 nodes (0.05 seconds)
Cbc0031I 3 added rows had average density of 4
Cbc0013I At root node, 3 cuts changed objective from -584902.22 to -584864.19 in
52 passes
Cbc0014I Cut generator 0 (Probing) - 0 row cuts average 0.0 elements, 0 column cu
ts (0 active)  in 0.002 seconds - new frequency is -100
Cbc0014I Cut generator 1 (Gomory) - 87 row cuts average 4.0 elements, 0 column cu
ts (0 active)  in 0.004 seconds - new frequency is 1
Cbc0014I Cut generator 2 (Knapsack) - 0 row cuts average 0.0 elements, 0 column c
uts (0 active)  in 0.001 seconds - new frequency is -100
Cbc0014I Cut generator 3 (Clique) - 0 row cuts average 0.0 elements, 0 column cut
s (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 4 (MixedIntegerRounding2) - 0 row cuts average 0.0 element
s, 0 column cuts (0 active)  in 0.002 seconds - new frequency is -100
Cbc0014I Cut generator 5 (FlowCover) - 0 row cuts average 0.0 elements, 0 column
cuts (0 active)  in 0.002 seconds - new frequency is -100
Cbc0010I After 0 nodes, 1 on tree, -584780 best solution, best possible -584864.1
9 (0.05 seconds)
Cbc0012I Integer solution of -584800 found by DiveCoefficient after 98 iterations
and 2 nodes (0.05 seconds)
Cbc0016I Integer solution of -584860 found by strong branching after 104 iteratio
ns and 2 nodes (0.05 seconds)
Cbc0001I Search completed - best objective -584860, took 104 iterations and 2 nod
es (0.05 seconds)
Cbc0032I Strong branching done 14 times (20 iterations), fathomed 1 nodes and fix
ed 0 variables
Cbc0035I Maximum depth 1, 1 variables fixed on reduced cost
Cuts at root node changed objective from -584902 to -584864
Probing was tried 52 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.002 seconds)
Gomory was tried 58 times and created 95 cuts of which 0 were active after adding
rounds of cuts (0.004 seconds)
Knapsack was tried 52 times and created 0 cuts of which 0 were active after addin
g rounds of cuts (0.001 seconds)
Clique was tried 52 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 52 times and created 0 cuts of which 0 were activ
e after adding rounds of cuts (0.002 seconds)
FlowCover was tried 52 times and created 0 cuts of which 0 were active after addi
ng rounds of cuts (0.002 seconds)
TwoMirCuts was tried 1 times and created 0 cuts of which 0 were active after addi
ng rounds of cuts (0.000 seconds)
ZeroHalf was tried 1 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.019 seconds)

Result - Optimal solution found

Objective value:                584860.00000000
Enumerated nodes:               2
Total iterations:               104
Time (CPU seconds):             0.05
Time (Wallclock seconds):       0.03
```

```
  Total time (CPU seconds):       0.05   (Wallclock seconds):       0.03
```

In [33]:

```
using JuMP, Cbc
m = Model(Cbc.Optimizer)
@variable(m, λ[1:3] >= 0, Int) # variables for each primal constraint

# constraints ensuring the correct relationship with each primal variable
# to guarantee an upper bound
@constraint(m, 2λ[1] + 5λ[2] + 0.25λ[3] >= 60)
@constraint(m, 3λ[1] + 5λ[2] + λ[3] >= 120)
@constraint(m, 3λ[1] + 10λ[2] + 2λ[3] >= 200)
@constraint(m, 5λ[1] + 15λ[2] + 3.5λ[3] >= 300)

# objective is to minimize the upper bound on the primal solution
@objective(m, Min, 12001λ[1] + 32000λ[2] + 5000λ[3] )

# solve this instance of the Top Brass dual
optimize!(m)

# print the dual model and solution
display(m)

println("dual variables are: ", value.(λ))
println("Optimal objective is: ", objective_value(m))
```

$$
\begin{aligned}
\min \quad & 12001\lambda_1 + 32000\lambda_2 + 5000\lambda_3 \\
\text{Subject to} \quad & 2\lambda_1 + 5\lambda_2 + 0.25\lambda_3 \geq 60.0 \\
& 3\lambda_1 + 5\lambda_2 + \lambda_3 \geq 120.0 \\
& 3\lambda_1 + 10\lambda_2 + 2\lambda_3 \geq 200.0 \\
& 5\lambda_1 + 15\lambda_2 + 3.5\lambda_3 \geq 300.0 \\
& \lambda_1 \geq 0.0 \\
& \lambda_2 \geq 0.0 \\
& \lambda_3 \geq 0.0 \\
& \lambda_1 \, integer \\
& \lambda_2 \, integer \\
& \lambda_3 \, integer
\end{aligned}
$$

```
dual variables are: [13.0, 4.0, 61.0]
Optimal objective is: 589013.0
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: May 23 2020

command line - Cbc_C_Interface -solve -quit (default strategy 1)
Continuous objective value is 584902 - 0.00 seconds
Cgl0003I 0 fixed, 3 tightened bounds, 0 strengthened rows, 0 substitutions
Cgl0004I processed model has 4 rows, 3 columns (3 integer (0 of which binary)) and 1
2 elements
Cbc0012I Integer solution of 611014 found by DiveCoefficient after 0 iterations and
0 nodes (0.00 seconds)
Cbc0012I Integer solution of 591015 found by DiveCoefficient after 57 iterations and
0 nodes (0.03 seconds)
Cbc0031I 2 added rows had average density of 3
Cbc0013I At root node, 2 cuts changed objective from 584902.22 to 587751.65 in 38 pa
sses
Cbc0014I Cut generator 0 (Probing) - 0 row cuts average 0.0 elements, 1 column cuts
(1 active)  in 0.001 seconds - new frequency is 1
```

```
Cbc0014I Cut generator 1 (Gomory) - 49 row cuts average 3.0 elements, 0 column cuts
(0 active)  in 0.002 seconds - new frequency is 1
Cbc0014I Cut generator 2 (Knapsack) - 0 row cuts average 0.0 elements, 0 column cuts
(0 active)  in 0.001 seconds - new frequency is -100
Cbc0014I Cut generator 3 (Clique) - 0 row cuts average 0.0 elements, 0 column cuts
(0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 4 (MixedIntegerRounding2) - 1 row cuts average 3.0 elements,
0 column cuts (0 active)  in 0.001 seconds - new frequency is -100
Cbc0014I Cut generator 5 (FlowCover) - 0 row cuts average 0.0 elements, 0 column cut
s (0 active)  in 0.002 seconds - new frequency is -100
Cbc0010I After 0 nodes, 1 on tree, 591015 best solution, best possible 587751.65 (0.
03 seconds)
Cbc0004I Integer solution of 589013 found after 70 iterations and 1 nodes (0.03 seco
nds)
Cbc0001I Search completed - best objective 589013, took 72 iterations and 2 nodes
(0.03 seconds)
Cbc0032I Strong branching done 6 times (7 iterations), fathomed 0 nodes and fixed 0
variables
Cbc0035I Maximum depth 0, 0 variables fixed on reduced cost
Cuts at root node changed objective from 584902 to 587752
Probing was tried 45 times and created 2 cuts of which 0 were active after adding ro
unds of cuts (0.002 seconds)
Gomory was tried 45 times and created 62 cuts of which 0 were active after adding ro
unds of cuts (0.003 seconds)
Knapsack was tried 38 times and created 0 cuts of which 0 were active after adding r
ounds of cuts (0.001 seconds)
Clique was tried 38 times and created 0 cuts of which 0 were active after adding rou
nds of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 38 times and created 1 cuts of which 0 were active a
fter adding rounds of cuts (0.001 seconds)
FlowCover was tried 38 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.002 seconds)
TwoMirCuts was tried 1 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.000 seconds)
ZeroHalf was tried 1 times and created 0 cuts of which 0 were active after adding ro
unds of cuts (0.010 seconds)

Result - Optimal solution found

Objective value:                589013.00000000
Enumerated nodes:               2
Total iterations:               72
Time (CPU seconds):             0.03
Time (Wallclock seconds):       0.03

Total time (CPU seconds):       0.03   (Wallclock seconds):       0.03
```

By comparison of two dual problem, willing to pay 13 dollars.

(c)

Estimate:
previous optimal solution:
x1 = 1866.9999999999998
x2 = 977.0
x3 = 1778.0

x4 = 0.0
since consstraints do not change, this solution is still feasible for the new problem, hence, the new max profit $z*_{NEW} \geq 60*x1+130*x2+215*x3+300*x4=621300$.

The new optimal profit:

In [34]:

```
using JuMP, Cbc
m = Model(Cbc.Optimizer)

@variable(m, x1 >= 0, Int)
@variable(m, x2 >= 0, Int)
@variable(m, x3 >= 0, Int)
@variable(m, x4 >= 0, Int)


@constraint(m, con1,   2*x1+ 3*x2+ 3*x3+ 5*x4 <= 12000)
@constraint(m, con2,  5*x1+ 5*x2+ 10*x3+ 15*x4 <= 32000)
@constraint(m, con3,   0.25*x1+ x2+ 2*x3+ 3.5*x4 <= 5000 )


@objective(m, Max, 60*x1+ 130*x2+ 215*x3+ 300*x4)              # maximize profit

# solve this instance of the Top Brass problem
optimize!(m)

# print out the full model and solution
display(m)

println("x1 = ", value(x1))
println("x2 = ", value(x2))
println("x3 = ", value(x3))
println("x4 = ", value(x4))

println("max profit will be \$", objective_value(m))
```

$$
\begin{aligned}
\max \quad & 60x1 + 130x2 + 215x3 + 300x4\\
\text{Subject to} \quad & 2x1 + 3x2 + 3x3 + 5x4 \leq 12000.0\\
& 5x1 + 5x2 + 10x3 + 15x4 \leq 32000.0\\
& 0.25x1 + x2 + 2x3 + 3.5x4 \leq 5000.0\\
& x1 \geq 0.0\\
& x2 \geq 0.0\\
& x3 \geq 0.0\\
& x4 \geq 0.0\\
& x1 integer\\
& x2 integer\\
& x3 integer\\
& x4 integer
\end{aligned}
$$

```
x1 = 1866.9999999999998
x2 = 977.0
x3 = 1778.0
x4 = 0.0
max profit will be $621300.0
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: May 23 2020

command line - Cbc_C_Interface -solve -quit (default strategy 1)
Continuous objective value is 621333 - 0.00 seconds
Cgl0004I processed model has 3 rows, 4 columns (4 integer (0 of which binary)) an
d 12 elements
```

```
Cbc0012I Integer solution of -621025 found by DiveCoefficient after 0 iterations
and 0 nodes (0.00 seconds)
Cbc0038I Full problem 3 rows 4 columns, reduced to 3 rows 3 columns
Cbc0012I Integer solution of -621300 found by DiveCoefficient after 6 iterations
and 0 nodes (0.03 seconds)
Cbc0031I 3 added rows had average density of 4
Cbc0013I At root node, 3 cuts changed objective from -621333.33 to -621300 in 6 p
asses
Cbc0014I Cut generator 0 (Probing) - 0 row cuts average 0.0 elements, 0 column cu
ts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 1 (Gomory) - 11 row cuts average 4.0 elements, 0 column cu
ts (0 active)  in 0.000 seconds - new frequency is 1
Cbc0014I Cut generator 2 (Knapsack) - 0 row cuts average 0.0 elements, 0 column c
uts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 3 (Clique) - 0 row cuts average 0.0 elements, 0 column cut
s (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 4 (MixedIntegerRounding2) - 0 row cuts average 0.0 element
s, 0 column cuts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 5 (FlowCover) - 0 row cuts average 0.0 elements, 0 column
cuts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0001I Search completed - best objective -621300, took 6 iterations and 0 nodes
(0.03 seconds)
Cbc0035I Maximum depth 0, 0 variables fixed on reduced cost
Cuts at root node changed objective from -621333 to -621300
Probing was tried 6 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.000 seconds)
Gomory was tried 6 times and created 11 cuts of which 0 were active after adding
rounds of cuts (0.000 seconds)
Knapsack was tried 6 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.000 seconds)
Clique was tried 6 times and created 0 cuts of which 0 were active after adding r
ounds of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 6 times and created 0 cuts of which 0 were active
after adding rounds of cuts (0.000 seconds)
FlowCover was tried 6 times and created 0 cuts of which 0 were active after addin
g rounds of cuts (0.000 seconds)
TwoMirCuts was tried 1 times and created 0 cuts of which 0 were active after addi
ng rounds of cuts (0.000 seconds)
ZeroHalf was tried 1 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.023 seconds)

Result - Optimal solution found

Objective value:                621300.00000000
Enumerated nodes:               0
Total iterations:               6
Time (CPU seconds):             0.03
Time (Wallclock seconds):       0.02

Total time (CPU seconds):       0.03   (Wallclock seconds):       0.02
```

(d)

Estimate:
x1 == 1866.9999999999998
x2 == 977.0
x3 == 1778.0

x4 == 0.0

Consider the constriants

2*x1+ 3*x2+ 3*x3+ 5*x4 = 11999

5*x1+ 5*x2+ 10*x3+ 15*x4 = 32000

0.25*x1+ x2+ 2*x3+ 3.5*x4 = 4999.75

If we decrease the labor availability by 1000 hours, the first contraint will be binding. Since the object function does not change, new optimal profit $z*_{NEW} \leq 60*x1+120*x2+200*x3+300*x4 = 584860$.

In [35]:

```
using JuMP, Cbc
m = Model(Cbc.Optimizer)

@variable(m, x1 >= 0, Int)
@variable(m, x2 >= 0, Int)
@variable(m, x3 >= 0, Int)
@variable(m, x4 >= 0, Int)


@constraint(m, con1,   2*x1+ 3*x2+ 3*x3+ 5*x4 <= 11000)
@constraint(m, con2,  5*x1+ 5*x2+ 10*x3+ 15*x4 <= 36000)
@constraint(m, con3,   0.25*x1+ x2+ 2*x3+ 3.5*x4 <= 5000 )


@objective(m, Max, 60*x1+ 120*x2+ 200*x3+ 300*x4)              # maximize profit

# solve this instance of the Top Brass problem
optimize!(m)

# print out the full model and solution
display(m)

println("x1 = ", value(x1))
println("x2 = ", value(x2))
println("x3 = ", value(x3))
println("x4 = ", value(x4))

println("max profit will be \$", objective_value(m))
```

$$
\begin{aligned}
\max \quad & 60x1 + 120x2 + 200x3 + 300x4\\
\text{Subject to} \quad & 2x1 + 3x2 + 3x3 + 5x4 \leq 11000.0\\
& 5x1 + 5x2 + 10x3 + 15x4 \leq 36000.0\\
& 0.25x1 + x2 + 2x3 + 3.5x4 \leq 5000.0\\
& x1 \geq 0.0\\
& x2 \geq 0.0\\
& x3 \geq 0.0\\
& x4 \geq 0.0\\
& x1 integer\\
& x2 integer\\
& x3 integer\\
& x4 integer
\end{aligned}
$$

```
x1 = 2152.0
x2 = 2.0
x3 = 2230.0
x4 = 0.0
max profit will be $575360.0
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: May 23 2020

command line - Cbc_C_Interface -solve -quit (default strategy 1)
Continuous objective value is 575385 - 0.00 seconds
Cgl0004I processed model has 3 rows, 4 columns (4 integer (0 of which binary)) an
d 12 elements
```

```
Cbc0012I Integer solution of -575180 found by DiveCoefficient after 0 iterations
and 0 nodes (0.00 seconds)
Cbc0038I Full problem 3 rows 4 columns, reduced to 3 rows 2 columns
Cbc0012I Integer solution of -575360 found by DiveCoefficient after 1 iterations
and 0 nodes (0.02 seconds)
Cbc0031I 1 added rows had average density of 4
Cbc0013I At root node, 1 cuts changed objective from -575384.62 to -575360 in 7 p
asses
Cbc0014I Cut generator 0 (Probing) - 0 row cuts average 0.0 elements, 10 column c
uts (10 active)  in 0.000 seconds - new frequency is 1
Cbc0014I Cut generator 1 (Gomory) - 2 row cuts average 4.0 elements, 0 column cut
s (0 active)  in 0.000 seconds - new frequency is 1
Cbc0014I Cut generator 2 (Knapsack) - 0 row cuts average 0.0 elements, 0 column c
uts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 3 (Clique) - 0 row cuts average 0.0 elements, 0 column cut
s (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 4 (MixedIntegerRounding2) - 0 row cuts average 0.0 element
s, 0 column cuts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 5 (FlowCover) - 0 row cuts average 0.0 elements, 0 column
cuts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0001I Search completed - best objective -575360, took 1 iterations and 0 nodes
(0.02 seconds)
Cbc0035I Maximum depth 0, 0 variables fixed on reduced cost
Cuts at root node changed objective from -575385 to -575360
Probing was tried 7 times and created 10 cuts of which 0 were active after adding
rounds of cuts (0.000 seconds)
Gomory was tried 7 times and created 2 cuts of which 0 were active after adding r
ounds of cuts (0.000 seconds)
Knapsack was tried 7 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.000 seconds)
Clique was tried 7 times and created 0 cuts of which 0 were active after adding r
ounds of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 7 times and created 0 cuts of which 0 were active
after adding rounds of cuts (0.000 seconds)
FlowCover was tried 7 times and created 0 cuts of which 0 were active after addin
g rounds of cuts (0.000 seconds)
TwoMirCuts was tried 1 times and created 0 cuts of which 0 were active after addi
ng rounds of cuts (0.000 seconds)
ZeroHalf was tried 1 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.019 seconds)

Result - Optimal solution found

Objective value:                575360.00000000
Enumerated nodes:               0
Total iterations:               1
Time (CPU seconds):             0.03
Time (Wallclock seconds):       0.02

Total time (CPU seconds):       0.03   (Wallclock seconds):       0.02
```

In [ ]:

In [ ]: