drivers_customers_problem

January 20, 2016

```
In [2]: %matplotlib inline
        import matplotlib.pyplot as plt

        from numpy.matlib import repmat
        import cvxpy as cvx
        import gurobipy as grb
        import scipy.optimize

        import networkx as nx
        import numpy as np
        import scipy as sp
        from scipy.sparse import rand
        from itertools import combinations
        from scipy.sparse import csr_matrix
```

0.0.1 Mixed Integer Problem

- i - orders, j drivers
- Let's $y_{ij} \in \{0, 1\}$ represent whether driver j delivers order i. The variable takes on value 1 if a delivery was made and 0 otherwise.
- Let's $x_j \in \{0, 1\}$ represent whether driver j is in game or not :)
- Objective is to minimize the total cost:

$$\begin{aligned} \underset{j}{\text{minimize}} \quad & \sum_j cost_j x_j \\ s.t. \quad & \sum_j y_{ij} = 1 \ \forall i \\ & \sum_i order_i * y_{ij} \leq c_j \ \forall j \\ & x_j \geq \frac{\sum_i y_{ij}}{nmb_{orders}} \end{aligned}$$

```
In [44]: def MIPSolver(orders, capacities, costs):

             nmb_drivers = len(capacities)

             Y = cvx.Bool(len(orders), nmb_drivers)
             X = cvx.Bool(nmb_drivers)

             objective = cvx.Minimize(costs.T * X)

             constraints = []
```

```
    for j in range(nmb_drivers):
        constraints.append(Y[:,j].T * orders <= capacities[j])
    for j in range(nmb_drivers):
        constraints.append(X[j] * len(orders) >=  cvx.sum_entries(Y[:,j]))
    for i in range(len(orders)):
        constraints.append(cvx.sum_entries(Y[i,:]) == 1)

    solution = cvx.Problem(objective, constraints)
    solution.solve()

    result_cost = solution.value

    return result_cost, Y.value, X.value


def MIPSolver_with_trash(orders, capacities, costs):

    cost_trash = costs.sum() + costs.max()
    costs = np.hstack((costs, cost_trash))
    capacity_trash = orders.sum()
    capacities = np.hstack((capacities, capacity_trash))

    nmb_drivers = len(capacities)

    Y = cvx.Bool(len(orders), nmb_drivers)
    X = cvx.Bool(nmb_drivers)

    objective = cvx.Minimize(costs.T * X)

    constraints = []

    for j in range(nmb_drivers):
        constraints.append(Y[:,j].T * orders <= capacities[j])
    for j in range(nmb_drivers):
        constraints.append(X[j] * len(orders) >=  cvx.sum_entries(Y[:,j]))
    for i in range(len(orders)):
        constraints.append(cvx.sum_entries(Y[i,:]) == 1)

    solution = cvx.Problem(objective, constraints)
    solution.solve()

    result_cost = solution.value

    return result_cost, Y.value, X.value
```

0.0.2 Test

In [61]:
```
np.random.seed(42)

nmb_drivers = 4
nmb_customers = 20

orders = np.random.randint(1, 3, nmb_customers)
```

```
        capacities = np.random.randint(1, 40, nmb_drivers)
        costs = np.random.sample(nmb_drivers)
```

In [63]:
```
        print 'costs: ', costs
        print 'capacities: ', capacities
        print 'orders: ', orders
```

```
[ 0.21233911  0.18182497  0.18340451  0.30424224]
[22  2 24 30]
[1 2 1 1 1 2 1 1 1 2 1 1 1 1 2 1 2 2 2 1]
```

In [62]:
```
        result, Y, X = MIPSolver(orders, capacities, costs)
        print "result cost:", result
        print "matrix orders - drivers: ", Y
```

```
result cost: 0.39574362053
matrix orders - drivers:  [[  1.00000000e+00   2.67450899e-12  -3.57569970e-12  -2.43784190e-12]
 [  1.00000000e+00   4.57606495e-12  -5.02125847e-12  -3.51021910e-12]
 [  3.35247957e-12   2.67450137e-12   1.00000000e+00  -2.43784952e-12]
 [  3.35247957e-12   2.67450137e-12   1.00000000e+00  -2.43784952e-12]
 [  3.35247957e-12   2.67450137e-12   1.00000000e+00  -2.43784952e-12]
 [  3.97333609e-12   4.57610990e-12   1.00000000e+00  -3.51017410e-12]
 [  3.35247957e-12   2.67450137e-12   1.00000000e+00  -2.43784952e-12]
 [  3.35247957e-12   2.67450137e-12   1.00000000e+00  -2.43784952e-12]
 [  3.35247957e-12   2.67450137e-12   1.00000000e+00  -2.43784952e-12]
 [  3.97333609e-12   4.57610990e-12   1.00000000e+00  -3.51017410e-12]
 [  3.35247957e-12   2.67450137e-12   1.00000000e+00  -2.43784952e-12]
 [  3.35247957e-12   2.67450137e-12   1.00000000e+00  -2.43784952e-12]
 [  3.35247957e-12   2.67450137e-12   1.00000000e+00  -2.43784952e-12]
 [  3.35247957e-12   2.67450137e-12   1.00000000e+00  -2.43784952e-12]
 [  3.97333609e-12   4.57610990e-12   1.00000000e+00  -3.51017410e-12]
 [  3.35247957e-12   2.67450137e-12   1.00000000e+00  -2.43784952e-12]
 [  3.97333609e-12   4.57610990e-12   1.00000000e+00  -3.51017410e-12]
 [  3.97333609e-12   4.57610990e-12   1.00000000e+00  -3.51017410e-12]
 [  3.97333609e-12   4.57610990e-12   1.00000000e+00  -3.51017410e-12]
 [  3.35247957e-12   2.67450137e-12   1.00000000e+00  -2.43784952e-12]]
```

Test: lack of drivers

In [82]:
```
np.random.seed(42)

        nmb_drivers = 2
        nmb_customers = 20

        orders = np.random.randint(1, 3, nmb_customers)
        capacities = np.random.randint(1, 10, nmb_drivers)
        costs = np.random.sample(nmb_drivers)
```

In [84]:
```
        result, Y, X = MIPSolver(orders, capacities, costs)
        print result
```

```
inf
```

In [83]:
```
        result, Y, X = MIPSolver_with_trash(orders, capacities, costs)

        print "result cost:", result
```

```
result cost: 2.77226234515
```