A Project For Police Patrol District Design

Group

Xuanzhou Chen (xchen793@wisc.edu)

Table of contents

1.Introduction
2.Data Generation
3.Three models
A. Balance the patrol center workload
a.Mathematical Model
b.Implementation
c.Result and Discussion
B. Prevent 7 murders without bias
a.Mathematical Model
b.Implementation
c.Result and Discussion
C. Intercept the criminal
a.Mathematical Model
b.Implementation
c.Result and Discussion

Introduction

My project will use three different mathematical models to implement the optimization problem for police patrol district design.

Due to limited police resource (given a determined graph set of police service platforms), it is necessary to establish a mathematical model based on the actual conditions and needs of the city to allocate the jurisdiction of each police service platform and distribute the police resources. In this optimization, I generate the position of police service platforms(patrol centers) by random data. By default, I set the number of patrol centers be 1/4 of total traffic roads. The basic principle of optimization is the variance and the minimum cases that each patrol center is required to handle.

We need to find math models to find out:
(1) Determine the area each patrol center should take control
(2) The way to distribute patrol group to each traffic road depending on how many cases in the traffic roads.

By default, the number of patrol center would be 1/4 of total traffic roads, we will randomly generate those patrol centers (data generation).

In this project I will use matching alogrithm, network flow model and linear programing to solve 3 particular problems that police officer will encounter in their work.

Problem 1: For a city given the coordinates of every traffic road and the coordinates of every patrol center, how to find out the distance between any two patrol centers and traffic roads, as well as which traffic roads will be covered in each patrol center? Could we get a solution on how to distribute the police force to each traffic road?

Problem 2: If 7 people are being murderd at the same time and there are only 10 police cars left in each patrol center , how to assign task for each patrol center so that without bias all 7 people will be saved as soon as possible?

Problem 3: Could we find out the minimum time to take down the criminal if he is reported escaping at some traffic road? Does it depend on the police's speed?

I will solve the 3 problems above using the following models:

1. Network flow model: each traffic road can be treat as a node in a network, and they will flow through each patrol center, what we're trying to do is to balance the flows to patrol center by minimizing the variance of each patrol center.

2. Linear programing model: We will use integer optimization model and minimax problem model to solve the decision variables that will provide us a solution that will garantee that each muder prevention will have the same proirity.

3. Matching algorithm: When we know the traffics roads that needed to be at each time interval we can solve for the matching between these traffic roads so that we can find out if it is possible to lock the criminal in the circle at time t.

Data Generation

Data Form

First an array expressing the locations of 40 traffic roads will be generated, then the adjacency matrix will be calculated from this array.

The city we will generate will be expressed as a 40x40 matrix which means it contains 40 traffic roads. Then the patrol center, the location where the criminal start to escape and the 7 traffic roads that the murder occurs will be generated as vectors.

Math Model

Some constraints on the random funtion is consider as followed:

The scales

Since the mean of random function seems to be 1/2 (uniformly distributed), Multinplying it by a fator (here is 5 as the lenght of the whole square city) makes it look more normal. To make these road concentarte around the 2.5*2.5 sqaure of the city we force 20 traffic roads to be in that range.

Implementation

In [1]:

```
import Pkg; Pkg.add("GraphRecipes")
import Pkg; Pkg.add("FixedPointNumbers")
using Random
using Graphs, NamedArrays, Clp, Combinatorics, Gurobi, JuMP, PyPlot
using GraphRecipes
using Plots
using FixedPointNumbers
Random.seed!(12345);
rng = MersenneTwister(12345);




lmap=rand(Float64,(40,2))

avg=0
rawmap=zeros(40,40)
for i in 1:40
    for j in 1:2
        if(i<20)
             lmap[i,j]=lmap[i,j]*5/2+5/4
        else
             lmap[i,j]*=5
        end
    end
end
cmap=zeros(40,40)
for i in 1:40
  avg+=lmap[i,1]
end

for i in 1:40
    for j in 1:40
      if(rand(Float64)<4/40)
            cmap[i,j]=1
            cmap[j,i]=1
      end
    end
end


for i in 1:40
    for j in 1:40
      if(cmap[i,j]==1)
            rawmap[i,j]=sqrt((lmap[i,1]-lmap[j,1])^2+(lmap[i,2]-lmap[j,2])^2)

      end
    end
end


xs=zeros(40)
ys=zeros(40)
for i in 1:40
    xs[i]=lmap[i,1]
    ys[i]=lmap[i,2]

end
ps=zeros(10)
```

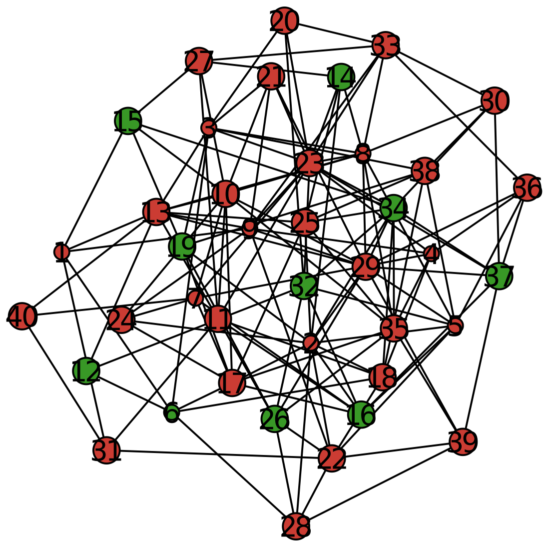
```
temp=shuffle(rng, Vector(1:40))
for i in 1:10
    ps[i]=temp[i]
end
colors= Array{RGB{FixedPointNumbers.Normed{UInt8,8}}}(undef,0)
po=0
for i in 1:40
    po=0
    for j in 1:10
        if (ps[j]==i)
            #println("pushing ",i)
           # println(colors)

            push!(colors,colorant"#389826")
            po=1
            continue
        end
    end

    if(po==0)
        push!(colors,colorant"#CB3C33")
    end
   # println(colors)




 end
println("The patrol centers are on the node:",ps)



finalCitymap=zeros(40,40)
for i in 1:40
    for j in 1:40
        finalCitymap[i,j]=rawmap[i,j]
    end
end
println("fianl city map is generated")
println(finalCitymap)


graphplot(rawmap,

          nodeshape=:circle,
          nodesize=0.1,

          curves=false,
          names = 1:40,
          fontsize = 10,
            nodecolor=colors,
          linecolor = :darkgrey
          )
```

```
   Updating registry at `C:\Users\10096\.julia\registries\General`
   Updating git-repo `https://github.com/JuliaRegistries/General.git`
  Resolving package versions...==============>]  100.0 %.0 %
   Updating `C:\Users\10096\.julia\environments\v1.3\Project.toml`
  [no changes]
   Updating `C:\Users\10096\.julia\environments\v1.3\Manifest.toml`
  [no changes]
  Resolving package versions...
   Updating `C:\Users\10096\.julia\environments\v1.3\Project.toml`
  [no changes]
   Updating `C:\Users\10096\.julia\environments\v1.3\Manifest.toml`
  [no changes]
The patrol centers are on the node:[16.0, 37.0, 32.0, 26.0, 6.0, 12.0, 15.0, 34.0,
14.0, 19.0]
fianl city map is generated
[0.0 0.0 0.0 0.0 0.0 1.3147356275325803 0.0 0.0 1.2402388679943672 0.0 0.0 0.0 0.2
4401429126976074 0.0 0.7478364763822882 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 0.0 0.0 1.9579309039937483 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0
 0.0 0.0 1.3636303031597445 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 2.7885301448905953 0.0
0.9400446941576964 1.7930494439117428 1.6039291528435249 0.6364353457511092 0.0 0.
0 1.862073581810069 0.0 4.288012291457969 1.9784950962967571 0.0 0.0 2.24173686652
93087 1.597957135839688 0.0 0.0 0.0 0.0 0.7341410645413724 0.0 0.0 0.0 0.0 0.0 0.
0; 0.0 0.0 0.0 0.0 0.0 2.0333082654878005 0.0 0.34491317016217665 0.0 0.0 0.0 0.0
 0.7889681410517357 0.0 0.0 0.0 0.0 0.0 1.7035188094019451 0.8291180654482114 3.60
15505024509156 0.0 3.385269418440739 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 0.0 0.0 0.0 2.937593559732527 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 2.327020810417148
 0.0 2.3083013092680633 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.8527341356655689 0.0 0.0
0.0 0.0 3.4741682045916162 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.159124869
4266383 2.5310480791965486 0.0 0.0 0.0 0.0; 0.0 1.3636303031597445 0.0 0.0 0.0 0.0
0.0 0.8198043967362628 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5992698377524339 0.0 0.0 0.0
 0.0 0.0 1.927953661652927 0.0 0.0 0.0 0.0 0.0 0.0 2.864023966783614 0.0 0.0 1.629
571427990713 0.0 0.0 0.0 1.580577079296287 0.0 2.0416924919611175 0.0 0.0; 1.31473
56275325803 0.0 2.0333082654878005 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.9069596282256
12 0.0 0.0 0.0 0.0 1.0012814902579408 0.6037021283237396 0.0 0.0 0.0 0.0 0.0 0.0
 0.0 0.0 0.0 2.919859997186114 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.
0 0.0 0.0 2.327020810417148 0.0 0.0 0.0 0.0 0.0 2.183127889099228 0.0 0.0 0.0 0.0
 0.0 1.018173388070414 1.7298734945040362 0.0 0.0 0.0 0.0 0.0 1.256144935386404 3.
4090189122798202 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 2.812
897407812699; 0.0 0.0 0.34491317016217665 0.0 0.8198043967362628 0.0 0.0 0.0 0.0
 0.9440780849031785 0.0 0.0 0.8983192839544929 2.0759708325489483 0.0 0.0 0.0 0.0
 0.0 0.0 0.0 0.0 3.3224411862299856 0.0 0.0 0.0 0.0 0.0 3.0725987371190566 1.94390
47763322155 0.0 0.0 1.703887200532162 0.0 1.2100557757338362 0.0 0.0 0.0 0.0 0.0;
 1.2402388679943672 0.0 0.0 2.3083013092680633 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.3
764678333235152 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.446214113163635 0.47046169457464304
 1.4576063234892183 3.689217102346471 0.0 0.0 0.0 0.0 2.015034694832162 0.0 0.0 0.
0 3.5673878715973486 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 2.183127
889099228 0.9440780849031785 0.0 0.0 1.6072523887062649 0.0 1.4209230260531562 0.0
1.685322568440766 0.0 0.4582114028727534 0.0 0.0 0.0 3.6731384749966183 0.0 0.0 2.
442098698336773 0.0 0.0 2.0998356755193197 0.0 3.6952799571286135 0.0 0.0 0.0 0.0
 0.0 1.9420573270477912 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 1.6072523887062649 0.0 0.0 1.2927239511977109 0.0 0.0 0.8661082774430194 0.0 0.15
927903594092685 0.0 0.0 0.0 0.0 1.4872427889501034 0.0 0.0 2.729295129269066 2.995
049782249853 0.0 0.0 0.0 1.1113572890269245 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0;
 0.0 0.0 0.0 0.0 0.0 1.906959628225612 0.0 0.0 0.0 0.0 0.0 0.0 1.0404730780815354
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.7593173
554839301 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.24401429126976074 0.0 0.7889681410517
357 0.0 0.0 0.0 0.0 0.8983192839544929 1.3764678333235152 1.4209230260531562 1.292
7239511977109 1.0404730780815354 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 1.3837376663782595 0.0 0.0 0.0 2.2817596068807573 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 2.0829009303565447; 0.0 2.7885301448905953 0.0 0.0 0.0 0.0 0.0 2.075970832
5489483 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 2.296217554982736
```

0.0 2.059256468914019 0.0 3.301798833327834 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0; 0.7478364763822882 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.68532256844
0766 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.1650323666253117 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 1.3805410185931153 0.0 0.0 0.0 0.0 0.0 0.0 1.1321446830199626 0.0 0.0 0.0 0.0 0.
0 0.0; 0.0 0.9400446941576964 0.0 0.0 0.5992698377524339 0.0 1.018173388070414 0.0
0.0 0.0 0.8661082774430194 0.0 0.0 0.0 0.0 0.0 1.0152601944997885 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0913769064009562 0.0 0.0 0.0 0.0 1.9501
025778009635 0.0 0.0 0.0; 0.0 1.7930494439117428 0.0 0.0 0.0 1.0012814902579408 1.
7298734945040362 0.0 0.0 0.4582114028727534 0.0 0.0 0.0 0.0 0.0 1.0152601944997885
0.0 0.0 1.8941877290055904 0.0 0.0 0.0 0.0 2.5788552160592175 1.0291908570165331
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.631765577604327 0.0 0.0 0.0 0.0 0.0; 0.0 1.
6039291528435249 0.0 1.8527341356655689 0.0 0.6037021283237396 0.0 0.0 0.0 0.0 0.1
5927903594092685 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.1582415822438792 0.0 0.
0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.041334173797933 0.0 0.0 0.0 1.732158698469
4666 0.0 0.0; 0.0 0.6364353457511092 1.7035188094019451 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 0.0 0.0 1.1650323666253117 0.0 1.8941877290055904 0.0 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 3.5728923831980306 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 3.0531996
119248523 0.0 0.0; 0.0 0.0 0.8291180654482114 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 2.897179104649767 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 2.064988240669575 2.251982357164593 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0
3.6015505024509156 0.0 0.0 0.0 0.0 0.0 1.446214113163635 3.6731384749966183 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.2307048850040625 0.0 0.0 0.0 0.0 0.
0 2.4402772297509427 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 1.8620735818
10069 0.0 0.0 1.927953661652927 0.0 0.0 0.0 0.47046169457464304 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 0.0 1.1582415822438792 0.0 0.0 0.0 0.0 0.0 0.0 0.0 3.831415497128407 0.0
3.726033628256204 0.0 0.0 0.5073778850872563 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.7702156
735759558 0.0; 0.0 0.0 3.385269418440739 3.4741682045916162 0.0 0.0 1.256144935386
404 3.3224411862299856 1.4576063234892183 0.0 1.4872427889501034 0.0 0.0 2.2962175
54982736 0.0 0.0 0.0 0.0 0.0 2.897179104649767 1.2307048850040625 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.4709927694301914 4.882464869999629 2.1254927153446292
0.0 0.0 3.3581053600222823 0.0 0.0 0.0; 0.0 4.288012291457969 0.0 0.0 0.0 0.0 3.4
090189122798202 0.0 3.689217102346471 2.442098698336773 0.0 0.0 0.0 0.0 0.0 0.0 2.
5788552160592175 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 1.9784950962967571 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.3837376663782595 2.059256468914019 0.0 0.0 1.0291908570165331 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 2.204733491468839 1.7
090220118836175 0.0 0.0 2.6637477043374505 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 2.729295129269066 0.0 0.0 0.0 0.0 0.0 0.0 0.0 3.5728923831980306 0.0 0.0
3.831415497128407 0.0 0.0 0.0 0.0 0.0 2.2097147745432077 4.93742514128189 0.0 0.0
3.6393625434821044 0.0 0.0 3.133563016474603 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 2.0998356755193197 2.995049782249853 0.0 0.0 3.301798833327834
1.3805410185931153 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.46567589977938295 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 2.2417368665293087 0.0 0.
0 0.0 2.919859997186114 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 3.726033628256204 0.0 0.0 0.0 2.2097147745432077 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 3.0983129250124644 0.0; 0.0 1.597957135839688 0.0 0.0 2.8640239667
83614 0.0 0.0 3.0725987371190566 2.015034694832162 3.6952799571286135 0.0 0.0 2.28
17596068807573 0.0 0.0 0.0 0.0 0.0 0.0 0.0 2.4402772297509427 0.0 0.0 0.0 0.0 4.93
742514128189 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.7164951709963479 0.0 2.732261134514851
4.199679633452012 0.0 3.586342075124794 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.943904
7763322155 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 3.5527754466590724 1.1811112093229368 0.0 0.0 2.1014807386
917056 1.64388213349605 0.0 0.0; 1.9579309039937483 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 0.0 1.1113572890269245 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.507377885087256
3 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 3.3961985888
56508; 0.0 0.0 0.0 0.0 1.629571427990713 0.0 0.0 0.0 0.0 0.0 0.0 0.759317355483930
1 0.0 0.0 0.0 1.0913769064009562 0.0 0.0 0.0 2.064988240669575 0.0 0.0 1.470992769
4301914 0.0 0.0 3.6393625434821044 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.7079105326603977
0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.703887200532162 3.56738787
15973486 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 2.251982357164593 0.0 0.0 4.88246
4869999629 0.0 0.0 0.0 0.0 0.46567589977938295 0.0 0.0 3.5527754466590724 0.0 0.0 0.0

0.0 0.0 3.938683677066273 0.0 0.0 0.0 0.0; 0.0 0.7341410645413724 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.1321446830199626 0.0 0.0 1.041334173797933 0.0
0.0 0.0 0.0 2.1254927153446292 0.0 2.204733491468839 0.0 0.0 0.0 1.71649517099634
79 1.1811112093229368 0.0 0.7079105326603977 0.0 0.0 0.9147265457034718 0.0 2.5852
277489263042 0.0 0.0 0.0; 0.0 0.0 0.0 1.1591248694266383 0.0 0.0 0.0 1.21005577573
38362 0.0 1.9420573270477912 0.0 0.0 0.0 0.0 0.0 0.0 1.631765577604327 0.0 0.0 0.0
0.0 0.0 0.0 0.0 1.7090220118836175 3.133563016474603 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.9147265457034718 0.0 0.0 0.0 0.0 2.3789009621823842 0.0; 0.0 0.0 0.0 2.53104807
91965486 1.580577079296287 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 2.732261134514851 0.0 0.0 0.0 3.93868367706627
3 0.0 0.0 0.0 2.5761927268572697 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.9501025778009635 0.0 0.0 0.0 0.0 0.0 0.0 3.358105360022
2823 0.0 0.0 0.0 0.0 0.0 4.199679633452012 2.1014807386917056 0.0 0.0 0.0 2.585227
7489263042 0.0 2.5761927268572697 0.0 0.0 1.6780513530829027 0.0; 0.0 0.0 2.937593
559732527 0.0 2.0416924919611175 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1.7321586984694666 3.0531996119248523 0.0 0.0 0.0 0.0 0.0 2.6637477043374505 0.0
0.0 0.0 0.0 1.64388213349605 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.77021567
35759558 0.0 0.0 0.0 0.0 0.0 3.0983129250124644 3.586342075124794 0.0 0.0 0.0 0.0
0.0 2.3789009621823842 0.0 1.6780513530829027 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.
0 2.812897407812699 0.0 0.0 0.0 0.0 0.0 2.0829009303565447 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 3.396198588856508 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0]

Out[1]:



Problem1

Analysis on the problem

We can think of every traffic road as vertex v_i,i $\in${the set of patrol center index}.

The police need to arrive at the traffic road where the incident happend within 3 mins with a 60mph police car implies that the each patrol center should lie in the 3 miles circl of each traffic road so that it is possible for them to arrive on time.

So we can frist figure out the shortest distance between each patrol center and each traffic road and represent this disdtance as a matrix $D_{i,j}$
Then we find out each traffic road is cover by what patrol centers by iterating through matrix $D_{i,j}$, and represent it as a matrix $C_{i,j}$ in which $C_{i,j} = 1$ means that ith traffic road is covered by jth patrol center.
Then we check if Any traffic road can't be reached by any patrol center within 5 minutes, then connect them to the closest patrol center.

Then we will get the incident arrayIn_j and size arraySi_j which represent how many cases is in each traffic road and the number of cases that can be handeled by each patrol center. Then we will try to lower the variance and overload of each patrol center by using networkflow optimizer.

Math Model

In order to determine the shortest path between any two traffic roads in the city from the adjacency matrix we need to use warshall algorithnm .

1.1 warshall algorithnm.

let $D_{i,j}$ be the shortest path from traffic road i to j, the idea is:

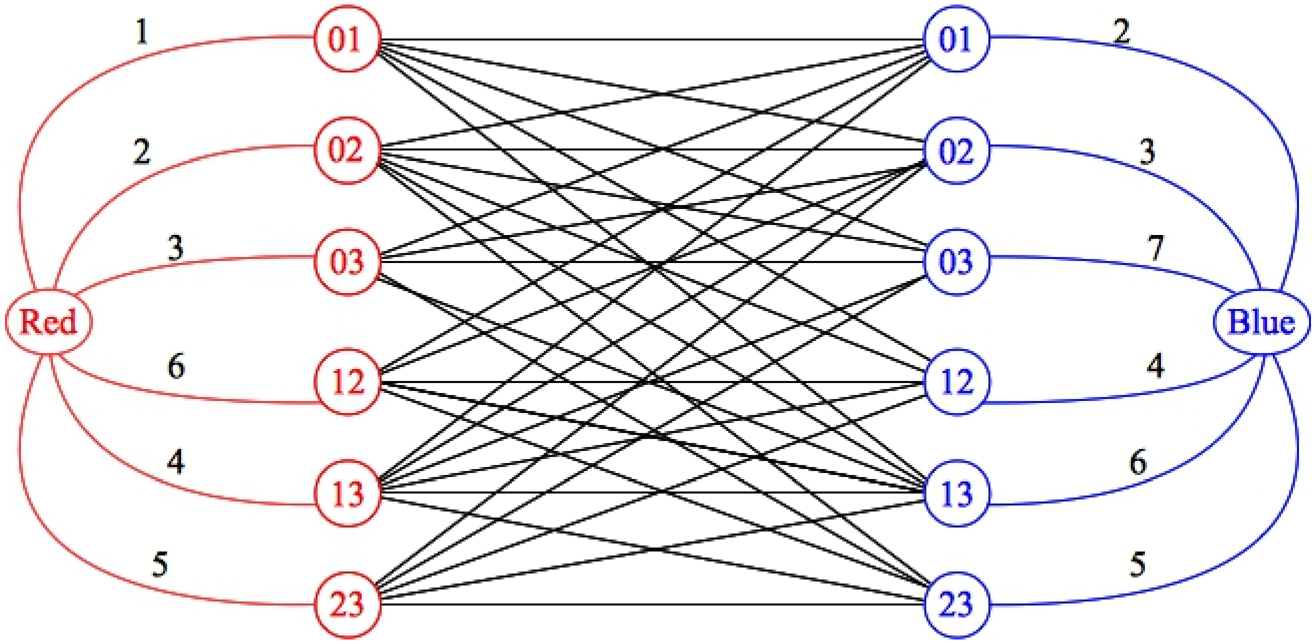
1)Get adjancecy matrix $A_{i,j}$ from the dataset , then initialize a new matrix that set the 0 entries to infinity.
2) for each pair of vertexs u and v, we will check if there is another vertex w such that $D_{u,w} + D_{w,v} > D_{u,v}$.

***pusedo code of this algorithmn:**

for k: 1 to n
    for i:=1 to n
        for j:=1 to n
            If $D_{i,j} > D_{i,k} + D_{k,j}$:
                $D_{i,j} = D_{i,k} + D_{k,j}$

1.2 create coverage matrix $C_{i,j}$ from $D_{i,j}$ and balance the overload of each police station

We first iterate through each entry of $D_{i,j}$, if $D_{i,j} < 3miles$ set $C_{i,j} = 1.$
The idea is use networkflow problem as a way to set the constraint



Let n be the total number of patrol centers , and let m be the total number of traffic roads,sp_j be the working speed of each patrol center when it is at its full capacity,cf be the full capacity of each patrol center.
The variable for optimizer solver would be $X_{i,j}$constriant would be

$$X_{i,j} = C_{i,j} * X_{i,j}$$
$$\sum_{j=1}^{n} X_{i,j} = flowin_i$$

$$\sum_{i=1}^{m} X_{i,j} <= maxload_j,$$

The incident array and patrol center maxload would be $flowin_i$ and $maxload_j$.

The variance objective function is computed as follow $z1 = min \quad \sum_{j=1}^{n} \frac{[\sum_{i=1}^{m} X_{i,j} - E(x)]^2}{n}$

Provided the comfortable load ,the overload objective function would be

$$z2 = min \quad \lfloor \frac{\sum_{i=1}^{m} X_{i,j}}{cf[j]} \rfloor * (\frac{\sum_{i=1}^{m} X_{i,j} - cf[j]}{sp[j]})$$

Implementation

In [2]:

```
import Pkg; Pkg.add("GraphRecipes")
import Pkg; Pkg.add("FixedPointNumbers")
import Pkg; Pkg.add("Ipopt")
import Pkg; Pkg.add("NLopt")
using Random
using Graphs, NamedArrays, Clp, Combinatorics, Gurobi, JuMP, PyPlot
```

```
using GraphRecipes
using Plots
using FixedPointNumbers


Random.seed!(1234);
rng = MersenneTwister(1234);

lmap=rand(Float64,(40,2))
#println(lmap)
avg=0
rawmap=zeros(40,40)
for i in 1:40
    for j in 1:2
        if(i<20)
             lmap[i,j]=lmap[i,j]*5/2+5/4
        else
             lmap[i,j]*=5
        end
    end
end
cmap=zeros(40,40)
for i in 1:40
  avg+=lmap[i,1]
end
#println(avg)
for i in 1:40
    for j in 1:40
      if(rand(Float64)<4/40)
            cmap[i,j]=1
            cmap[j,i]=1
      end
    end
end
 #println(cmap)

for i in 1:40
    for j in 1:40
      if(cmap[i,j]==1)
            rawmap[i,j]=sqrt((lmap[i,1]-lmap[j,1])^2+(lmap[i,2]-lmap[j,2])^2)

      end
    end
end


xs=zeros(40)
ys=zeros(40)
for i in 1:40
    xs[i]=lmap[i,1]
    ys[i]=lmap[i,2]

end
ps=zeros(10)
temp=shuffle(rng, Vector(1:40))
for i in 1:10
    ps[i]=temp[i]
end
colors= Array{RGB{FixedPointNumbers.Normed{UInt8,8}}}(undef,0)
po=0
for i in 1:40
```

```
    po=0
    for j in 1:10
        if (ps[j]==i)

            push!(colors,colorant"#389826")
            po=1
            break
        end
    end
    if(po==0)
        push!(colors,colorant"#CB3C33")
    end




 end
println("The patrol centers are on the node:",ps)



finalCitymap=zeros(40,40)
for i in 1:40
    for j in 1:40
        finalCitymap[i,j]=rawmap[i,j]
    end
end
#println("fianl city map is generated")
#println(finalCitymap)


plt=graphplot(rawmap,

          nodeshape=:circle,
          nodesize=0.1,

          curves=false,
          names = 1:40,
          fontsize = 10,
            nodecolor=colors,
          linecolor = :darkgrey
          )


n=40

for i in 1:n
    for j in 1:n
        if (i==j)
            rawmap[i,i]=0

        elseif ( rawmap[i,j]==0)
            rawmap[i,j]=10000
        end
    end
end
#println(rawmap)

for k in 1:n
    for i in 1:n
       for j in 1:n
```

```
            if(rawmap[i,j]>rawmap[i,k]+rawmap[k,j])
                  rawmap[i,j]=rawmap[i,k]+rawmap[k,j]
              end

          end
      end
end

#println(rawmap)
covermap=zeros(10,40)
bcovermap=zeros(10,40)
for i in 1:10
       for j in 1:40
          if(rawmap[Int(ps[i]),j]<3)
               covermap[i,j]= rawmap[Int(ps[i]),j]
               bcovermap[i,j]=1
            end

        end
end
#println(covermap)
#println(bcovermap)
for i in 1:10

    for j in 1:40
        if(j in ps)
           bcovermap[i,j]=0
        end
    end
end
for i in 1:10
    println("patrol center", i ,"covers:")
    for j in 1:40
        if(bcovermap[i,j]==1)
            println(j,",")
        end
    end
end


printcovermap=zeros(40,40)

for i in 1:10
    for j in 1:40
          if(covermap[i,j]>0)
             printcovermap[i,j]=covermap[i,j]
           end
    end
end




psl=8*rand(Float64,10)
inflow=2*rand(Float64,40)
expx=0
tdeg=zeros(40)
pl=zeros(10)
```

```
println("The crime cases in each traffic road is:",inflow)
println("The capacity of each patrol center is:",psl)
#println(pl)
#println(expx)
#println(bcovermap)


#println("Before optimization the variance is:",sum((pl[i]-expx)^2/10 for i in 1:10
 ))

m = Model(Gurobi.Optimizer)
@variable(m,  x[1:10,1:40])

@constraint(m,  con1[ i in 1:10,  j in 1:40],x[i,j]==bcovermap[i,j]*x[i,j])
@constraint(m,  con4[ i in 1:10,  j in 1:40],x[i,j]>=0)
@constraint(m,  con3[ i in 1:10],sum(x[i,j] for j in 1:40 )/inflow[i]>=0.8)
@constraint(m,  con2[ j in 1:40],sum(x[i,j] for i in 1:10 ) <= inflow[j])
println("1")

@constraint(m,  con5[   i in 1:10],sum(x[i,j] for j in 1:40) <= psl[i])
@objective(m,  Min,  sum((sum(x[i,j] for j in 40 )-sum(x[n,m]/10 for n in 1:10  for m
in 1:40))^2 for i in 1:10  )/10)

optimize!(m)
println("The how much cases should be handled by each patrol center:")
println(value.(x))
resultmap=zeros(40,40)
for i in 1:40
    for j in 1:40
        for k in 1:10
            if(i == ps[k])
                  resultmap[i,j]=value.(x)[k,j]
                resultmap[j,i]=value.(x)[k,j]
            end
        end
    end
end

graphplot(resultmap,

          nodeshape=:circle,
          nodesize=0.1,
           linesize=0.001,
          curves=false,
          names = 1:40,
          fontsize = 10,
            nodecolor=colors,
          linecolor = :darkgrey
          )
```

```
 Resolving package versions...
  Updating `C:\Users\10096\.julia\environments\v1.3\Project.toml`
 [no changes]
  Updating `C:\Users\10096\.julia\environments\v1.3\Manifest.toml`
 [no changes]
 Resolving package versions...
  Updating `C:\Users\10096\.julia\environments\v1.3\Project.toml`
 [no changes]
  Updating `C:\Users\10096\.julia\environments\v1.3\Manifest.toml`
 [no changes]
 Resolving package versions...
  Updating `C:\Users\10096\.julia\environments\v1.3\Project.toml`
 [no changes]
  Updating `C:\Users\10096\.julia\environments\v1.3\Manifest.toml`
 [no changes]
 Resolving package versions...
  Updating `C:\Users\10096\.julia\environments\v1.3\Project.toml`
 [no changes]
  Updating `C:\Users\10096\.julia\environments\v1.3\Manifest.toml`
 [no changes]
The patrol centers are on the node:[28.0, 38.0, 31.0, 32.0, 33.0, 18.0,
 37.0, 9.0, 27.0, 14.0]
patrol center1covers:
4,
5,
6,
10,
11,
13,
15,
24,
25,
35,
patrol center2covers:
3,
7,
17,
40,
patrol center3covers:
3,
6,
8,
11,
15,
20,
36,
39,
patrol center4covers:
2,
15,
40,
patrol center5covers:
2,
7,
8,
10,
11,
12,
13,
30,
35,
```

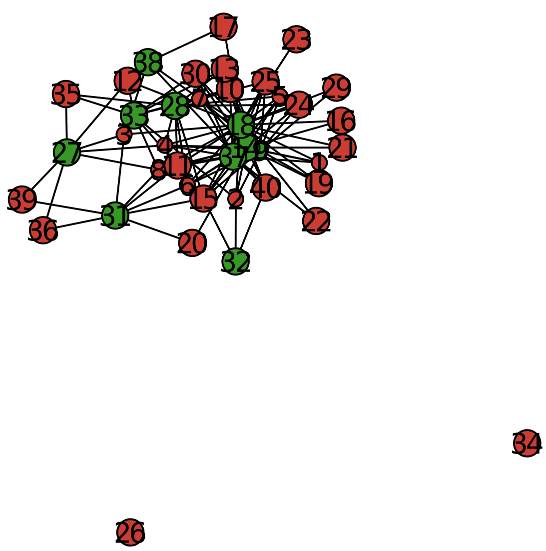
```
patrol center6covers:
1,
2,
3,
4,
5,
6,
7,
8,
10,
11,
12,
13,
15,
16,
17,
19,
21,
23,
24,
25,
29,
30,
40,
patrol center7covers:
1,
4,
5,
6,
7,
8,
10,
11,
15,
22,
25,
30,
40,
patrol center8covers:
1,
2,
4,
5,
6,
7,
8,
10,
11,
13,
15,
19,
22,
24,
25,
29,
patrol center9covers:
3,
4,
8,
12,
35,
```

```
36,
39,
patrol center10covers:
1,
2,
4,
5,
6,
7,
8,
10,
11,
13,
15,
16,
19,
20,
21,
24,
25,
30,
40,
The crime cases in each traffic road is:[0.5718276446950155, 1.384517473
8173975, 0.8503734992636205, 1.3824965879279967, 0.31840565555037603, 0.
598524309133202, 1.7202321631674384, 0.41512443018434775, 0.598903111285
7242, 0.5331078354561005, 1.4931418723189425, 0.13137181604552417, 0.436
8628637867964, 1.236669749591039, 0.6753083494709444, 0.935712347575751
8, 1.30356827784095, 0.4885013663970357, 0.3325931135571425, 1.798691021
4784118, 1.9923329033975747, 1.3971925744414704, 0.4026703405981906, 0.1
2945863613795705, 1.0468071410053366, 1.2982245691005438, 0.544502294215
4774, 0.2707600007581368, 0.5285204638442078, 0.830476415193218, 0.66205
90788831353, 0.9881328537009058, 0.859528360179111, 0.535492496810166,
 0.6291222809468517, 1.5607530921367947, 0.8761364795223496, 1.272765171
4838593, 1.8708374093866027, 0.6233598069259565]
The capacity of each patrol center is:[4.0270087667610195, 4.46637556868
2446, 3.9194505340182513, 6.734265788704446, 5.780587148554769, 1.086455
2901487912, 4.030355130748989, 3.019735970169796, 3.7698788113168007, 2.
4436935880757478]
Academic license - for non-commercial use only
1
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (win64)
Optimize a model with 860 rows, 400 columns and 1888 nonzeros
Model fingerprint: 0x7ea5d655
Model has 76300 quadratic objective terms
Coefficient statistics:
  Matrix range     [6e-01, 3e+00]
  Objective range  [0e+00, 0e+00]
  QObjective range [2e-02, 2e-01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e-01, 7e+00]
Presolve removed 823 rows and 288 columns
Presolve time: 0.09s
Presolved: 37 rows, 121 columns, 232 nonzeros
Presolved model has 5793 quadratic objective terms
Ordering time: 0.00s

Barrier statistics:
 Free vars  : 5
 AA' NZ     : 1.710e+02
 Factor NZ  : 7.030e+02
```

```
 Factor Ops : 1.519e+04 (less than 1 second per iteration)
 Threads    : 1

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Tim
e
   0   3.34177392e+05 -3.58767858e+05  2.67e+04 4.28e-06  1.00e+06     0
s
   1   1.33097958e+03 -5.18470985e+04  9.00e+02 1.44e-07  3.52e+04     0
s
   2   3.43629988e+00 -4.87967714e+04  2.38e+00 3.80e-10  3.97e+02     0
s
   3   3.15580630e+00 -5.01093327e+03  5.23e-02 8.37e-12  3.21e+01     0
s
   4   3.13652729e+00 -1.83202866e+01  1.60e-04 2.53e-14  1.35e-01     0
s
   5   1.85386452e+00 -4.93526712e+00  3.48e-05 5.55e-15  4.27e-02     0
s
   6   1.30405298e+00 -3.49130221e+00  1.85e-05 3.00e-15  3.02e-02     0
s
   7   5.99417399e-01 -3.20087402e-01  1.99e-06 4.44e-16  5.78e-03     0
s
   8   3.95517964e-01  5.42350884e-02  2.06e-07 5.55e-17  2.15e-03     0
s
   9   3.87906517e-01  3.02294484e-01  9.85e-09 1.86e-17  5.38e-04     0
s
  10   3.74293083e-01  3.59125744e-01  9.10e-15 5.55e-17  9.54e-05     0
s
  11   3.73004159e-01  3.72905736e-01  2.22e-16 1.37e-17  6.19e-07     0
s
  12   3.72996753e-01  3.72996654e-01  8.88e-16 3.00e-17  6.19e-10     0
s
  13   3.72996745e-01  3.72996745e-01  4.44e-16 1.96e-17  6.20e-13     0
s

Barrier solved model in 13 iterations and 0.10 seconds
Optimal objective 3.72996745e-01

The how much cases should be handled by each patrol center:
[0.0 0.0 0.0 0.04995982177453549 0.04267315843344367 0.05173887941759819
4 0.0 0.0 0.0 0.048497143138645 0.050314067584685734 0.0 0.0487213153766
15996 0.0 0.03406229431360458 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.03155941
9514259786 0.051568651742917054 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.04
8367364464713285 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.2903549509558251 0.0 0.0
0.0 0.3898619574512102 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.30272510915
92426 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 0.0 0.0 0.0 0.1246719614919219; 0.0 0.0 0.07295210025834385 0.
0 0.0 0.07397948431137073 0.0 0.054566918951322846 0.0 0.0 0.09365083832
117115 0.0 0.0 0.0 0.05115433151716762 0.0 0.0 0.0 0.0 0.121119536562572
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.1048306942
9049104 0.0 0.0 0.10804489520298799 0.0; 0.0 0.6285029625408969 0.0 0.0
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.35282234630806203 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 0.0 0.0 0.0 0.12467196149766638; 0.0 0.022066153050975052 0.0 0.0 0.0
 0.0 0.030180228279870644 0.027174126726649723 0.0 0.030146511831760422
 0.030997507078621735 0.021715863734876724 0.03021172286003626 0.0 0.0
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0319573536966
4022 0.0 0.0 0.0 0.0 0.030275057187088394 0.0 0.0 0.0 0.0 0.0; 0.0156003
69837172659 0.013917640031387138 0.015320374330892462 0.0170751292431059
2 0.01348080972228629 0.014916958485558345 0.016631080285450708 0.013472
378974237989 0.0 0.014490419237223495 0.016698320860869195 0.01204791321
```

```
330856 0.014397782073547333 0.0 0.013436195212943466 0.02164189493542492
0.018819844221892357 0.0 0.014974172280670056 0.0 0.02784974366935396 0.
0 0.017506896305207673 0.011844951966559287 0.016705293122827803 0.0 0.0
0.0 0.017118793745660825 0.016200524324639646 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 0.12467196123471622; 0.11314487182800842 0.0 0.0 0.12808959175
921372 0.07312534782050178 0.10098726427270852 0.12412320241443348 0.071
75511743814768 0.0 0.09312833044268784 0.1287194025181583 0.0 0.0 0.0 0.
062115486953338626 0.0 0.0 0.0 0.0 0.0 0.0 0.10839479620193397 0.0 0.0 0.
12638423552544495 0.0 0.0 0.0 0.0 0.12154612188509235 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 0.0 0.0 0.12467196147869823; 0.022309768705775553 0.0183438792
1648661 0.0 0.022893691098079705 0.018381763214187816 0.0206293306690669
2 0.022441473261306827 0.018334155595750714 0.0 0.019893745170375333 0.0
22597626445127347 0.0 0.019802008122230314 0.0 0.017789148713879852 0.0
 0.0 0.0 0.021737070999648334 0.0 0.0 0.025378706699193730 0.0 0.01551297
3696365772 0.022905510892345674 0.0 0.0 0.0 0.023148691656758764 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.06947377598188598 0.08241
499421751816 0.0 0.0 0.0 0.05542324109747911 0.0 0.0 0.0 0.0348606785331
2425 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.07262728183077453 0.08236011912984949 0.0 0.0 0.08
196239824255315 0.0; 0.016134853163132245 0.014096333420246660 0.0 0.0168
34298028327352 0.014276393353390726 0.015676727966681913 0.0164271257689
4296 0.014273725550338540 0.0 0.015280092299759372 0.016507875993881450 0.
0 0.015197369257451873 0.0 0.013819642219858505 0.020460747941905087 0.0
0.0 0.01579816047612364 0.024348560407608365 0.02707256500396468 0.0 0.0
0.012512778949511716 0.016734630951399567 0.0 0.0 0.0 0.0 0.016362426399
948456 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.12467196121889916]
```

Out[2]:



Result and Interpretation

1. **The row of matrix represent how much patrol force from i^{th} row should be sent to j^{th} traffic road.**

2. **They should be integer in the real world, so more investigation will be need. Here a better optimization solution is to further use the integer programming.**

3. **The company owner should build their store more evenly so some store won't sold out their trading cards and no store should be over stocked.**

Problem 2

Math Model

We can think of this problem as an integer optimization problem with decision variables.

$$X_{i,j} = \begin{cases} 1, & \text{if i\^{}\{th\} traffic road should be handled by j\^{}\{th\} patrol center} \\ 0, & \text{if } i^{th} \text{ traffic road should not be handled by } j^{th} \text{ patrol center} \end{cases}$$

And since one patrol center can only bloack one traffic road, and each traffic road only needs to be blocked by exactly one patrol center. These are two constraints we should consider in modeling.

What we need to do here is to extract the submatrix of problem 1 $C_{i,j}$ to represent the distance between n patrol centers and m traffic roads that need to be blocked.

Solution

In [3]:

```
using Random
using Graphs, NamedArrays, Clp, Combinatorics, Gurobi, JuMP, PyPlot
using GraphRecipes
using Plots
using FixedPointNumbers
using Ipopt
using NLopt

Random.seed!(1234);
rng = MersenneTwister(1234);
####################################################################
###################################################################
#      function to fint the shortest path between 2 traffic road
####################################################################
###################################################################
function Dijkstra(src,mat)
    visited=zeros(40)
    visited[src]=1
    dist=zeros(40)
    pre=zeros(40)
    minci=0
    for i in 1:40
    dist[i]=mat[src,i]
```

```
        pre[i]=src
    end


    for i in 1:40
       minc=10000000
        for j in 1:40
            if(visited[j]==0&&dist[j]<minc)
                minci=j
                minc=dist[j]
            end
        end
        visited[minci]=1
        for j in 1:40
           if(visited[j]==0&&mat[minci, j]!=10000000&&minc+mat[minci, j]<dist[j])
                dist[j]=minc+mat[minci, j]
                pre[j]=minci
            end
        end
    end


    return pre
end

function routea2b(a, b, mat)
   pre= Dijkstra(a, mat)
    i=Int(b)
    routeorder=[]
    print("route from ", a, " to ", b, " :")
    while(i!=a)
        push!(routeorder, i)
        i=Int(pre[i])
    end
    print(a, "->")

    for i in Iterators.reverse(1:length(routeorder)-1)
       print(routeorder[i], "->")
    end

end
######################################################################
#####################################################################
                   #generating data for the problem
######################################################################
#####################################################################


lmap=rand(Float64, (40, 2))
#println(lmap)
avg=0
rawmap=zeros(40, 40)
for i in 1:40
    for j in 1:2
        if(i<20)
             lmap[i, j]=lmap[i, j]*5/2+5/4
        else
             lmap[i, j]*=5
        end
    end
end
```

```
cmap=zeros(40,40)
for i in 1:40
  avg+=lmap[i,1]
end
println(avg/40)
for i in 1:40
    for j in 1:40
      if(rand(Float64)<4/40)
            cmap[i,j]=1
            cmap[j,i]=1
      end
    end
end
 #println(cmap)

for i in 1:40
    for j in 1:40
      if(cmap[i,j]==1)
            rawmap[i,j]=sqrt((lmap[i,1]-lmap[j,1])^2+(lmap[i,2]-lmap[j,2])^2)

      end
    end
end


xs=zeros(40)
ys=zeros(40)
for i in 1:40
    xs[i]=lmap[i,1]
    ys[i]=lmap[i,2]

end
ps=zeros(10)
temp=shuffle(rng, Vector(1:40))
for i in 1:10
    ps[i]=temp[i]
end


lockRoad=zeros(7)
for i in 1:7
    lockRoad[i]=temp[10+i]
end

colors= Array{RGB{FixedPointNumbers.Normed{UInt8,8}}}(undef,0)
po=0
for i in 1:40
    po=0
    for j in 1:10
        if (ps[j]==i)

            push!(colors,colorant"#389826")
            po=1
            break
        end
    end
    if(po==0)
        push!(colors,colorant"#CB3C33")
    end
```

```
 end
println("The patrol centers are on the node:",ps)




adj=zeros(40,40)


for i in 1:n
    for j in 1:n
        if (i==j)
            rawmap[i,i]=0

        elseif ( rawmap[i,j]==0)
            rawmap[i,j]=10000
        end
        adj[i,j]=rawmap[i,j]
    end
end
#println(rawmap)

for k in 1:n
    for i in 1:n
       for j in 1:n
          if(rawmap[i,j]>rawmap[i,k]+rawmap[k,j])
                rawmap[i,j]=rawmap[i,k]+rawmap[k,j]
            end

        end
    end
end

#println(rawmap)
covermap=zeros(10,40)
bcovermap=zeros(10,40)
for i in 1:10
       for j in 1:n
          if(rawmap[Int(ps[i]),j]<3)
               covermap[i,j]= rawmap[Int(ps[i]),j]
               bcovermap[i,j]=1
            end

        end
end
#println("covermap",covermap)
#println(bcovermap)
for i in 1:10

    for j in 1:40
        if(j in ps)
           bcovermap[i,j]=0

        end
    end
end

lockMap=zeros(10,7)
for i in 1:10
```

```
    for j in 1:7
        lockMap[i,j]=rawmap[Int(ps[i]),Int(lockRoad[j])]
     end
end
#println(lockRoad)
println("murder locations",lockRoad)
######################################################################
#####################################################################
                   #solution of the problem
######################################################################
#####################################################################
m = Model(Gurobi.Optimizer)
@variable(m, x[1:10,1:7],Bin)

@variable(m,r)
@constraint(m, con2[ i in 1:10 , j in 1:7],r>=lockMap[i,j]*x[i,j])
@constraint(m, con1[ i in 1:10],sum(x[i,j] for j in 1:7 )<=1)
@constraint(m, con4[ j in 1:7],sum(x[i,j] for i in 1:10)==1)

@objective(m, Min, r)
@time(optimize!(m))
println(value.(x))

for i in 1:10
   for j in 1:7
        if(value.(x)[i,j]==1)

            routea2b(Int(ps[i]),Int(lockRoad[j]),adj)

        end
    end
    println()
end
```

```
2.491098786608659

WARNING: using NLopt.optimize! in module Main conflicts with an existing
identifier.
```

```
The patrol centers are on the node:[28.0, 38.0, 31.0, 32.0, 33.0, 18.0,
37.0, 9.0, 27.0, 14.0]
murder locations[39.0, 30.0, 34.0, 17.0, 5.0, 15.0, 6.0]
Academic license - for non-commercial use only
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (win64)
Optimize a model with 87 rows, 71 columns and 280 nonzeros
Model fingerprint: 0xdbe669c7
Variable types: 1 continuous, 70 integer (70 binary)
Coefficient statistics:
  Matrix range     [9e-01, 7e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 4.5836477
Presolve removed 9 rows and 9 columns
Presolve time: 0.01s
Presolved: 78 rows, 62 columns, 244 nonzeros
Variable types: 1 continuous, 61 integer (61 binary)

Root relaxation: objective 1.062601e+00, 76 iterations, 0.01 seconds

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

     0     0    1.38328    0   20    4.58365    1.38328  69.8%     -
0s
H    0     0                       4.0729796    1.38328  66.0%     -
0s
H    0     0                       3.9474676    1.38328  65.0%     -
0s
     0     0    1.38328    0   20    3.94747    1.38328  65.0%     -
0s

Explored 1 nodes (123 simplex iterations) in 0.06 seconds
Thread count was 8 (of 8 available processors)

Solution count 3: 3.94747 4.07298 4.58365

Optimal solution found (tolerance 1.00e-04)
Best objective 3.947467604050e+00, best bound 3.947467604050e+00, gap 0.
0000%
  2.432449 seconds (2.16 M allocations: 108.432 MiB, 2.94% gc time)
[-0.0 -0.0 0.0 -0.0 0.0 -0.0 -0.0; -0.0 -0.0 -0.0 1.0 0.0 -0.0 -0.0; -0.
0 -0.0 0.0 -0.0 -0.0 -0.0 -0.0; 0.0 -0.0 0.0 0.0 0.0 -0.0 -0.0; -0.0 1.0
-0.0 0.0 -0.0 -0.0 -0.0; -0.0 -0.0 0.0 -0.0 1.0 -0.0 -0.0; -0.0 -0.0 0.0
-0.0 -0.0 -0.0 1.0; -0.0 -0.0 1.0 -0.0 -0.0 -0.0 -0.0; 1.0 -0.0 -0.0 -0.
0 -0.0 -0.0 -0.0; -0.0 -0.0 0.0 -0.0 -0.0 1.0 -0.0]

route from 38 to 17 :38->


route from 33 to 30 :33->
route from 18 to 5 :18->6->5->
route from 37 to 6 :37->6->
route from 9 to 34 :9->34->
route from 27 to 39 :27->39->
route from 14 to 15 :14->15->
```

Result and Interpretation

In this case, if there are 7 people are murdered, the minimum total distance is 3.947467604050e+00. We can obtain 7 routes from the output are
route from 38 to 17 :38->
route from 33 to 30 :33->
route from 18 to 5 :18->6->5->
route from 37 to 6 :37->6->
route from 9 to 34 :9->34->
route from 27 to 39 :27->39->
route from 14 to 15 :14->15->

Problem3

Math model

Since the criminal is escaping we need to calculte the possible area the criminal could be by drawing the cricle with $r_c = v_c * t$ where v is the escaping speed of the criminal and t is the time the criminal has been escaping.

Then we needs to iterate through the traffic roads in the row of the traffic road where the criminal start to escaping and add it to the blocking set B_i.

Then we need to calculate the police reaction circles with $r_p = v_p * y$, so that we can iterate through the patrol center to construct possible edges from set of patrol center P_i to B_i and represent this bipartite graph using a matrix $U_{i,j}$.

Then then we iterate through each entry of patrol center to traffic road matrix $T_{i,j}$ which contains the distance between each patrol center and each trafficroad.Set $U_{i,j} = T_{i,j}$ if these 2 condition is satisfied:
1)$T_{i,j} \leq r_p$
2) $i \in B_i$

Then We need to find a matching M_i in the bipartite graph$U_{i,j}$.

3.1 matching model in bipartite graph

We will use hungry algorithnm to find the matching in $U_{i,j}$.The idea of this algorithmn is that :.

1)If you can find an augmented path in bipartite graph, there is a better mathicing that is closer to perfect match than the current path.

2)The subgraph don't have augmented graph implies that there is not agumneted grap

pusedo code

```
U[n][m]
used[m]
r21[n]
main:
  for :i to n:
     initialize used
     if find(i) all+=1

find(x):
    for: i to m:
            if(U[x][i]==true&&used[i]==false):
                used[i]=1
                if(r21[i]==0||find(r21)):
                    r21=x
                    return true

  return false
```

Solution

In [4]:

```
using Random
using Graphs, NamedArrays, Clp, Combinatorics, Gurobi, JuMP, PyPlot
using GraphRecipes
using Plots
using FixedPointNumbers

######################################################################
#####################################################################
                   #generating data for the problem
######################################################################
#####################################################################
Random.seed!(1234);
rng = MersenneTwister(1234);

lmap=rand(Float64,(40,2))

avg=0
rawmap=zeros(40,40)
for i in 1:40
    for j in 1:2
        if(i<20)
             lmap[i,j]=lmap[i,j]*5/2+5/4
        else
             lmap[i,j]*=5
        end
    end
end
cmap=zeros(40,40)
for i in 1:40
  avg+=lmap[i,1]
end

for i in 1:40
    for j in 1:40
      if(rand(Float64)<4/40)
            cmap[i,j]=1
            cmap[j,i]=1
      end
    end
```

```
end
 #println(cmap)

for i in 1:40
    for j in 1:40
      if(cmap[i,j]==1)
            rawmap[i,j]=sqrt((lmap[i,1]-lmap[j,1])^2+(lmap[i,2]-lmap[j,2])^2)

      end
    end
end


xs=zeros(40)
ys=zeros(40)
for i in 1:40
    xs[i]=lmap[i,1]
    ys[i]=lmap[i,2]

end
ps=zeros(10)
temp=shuffle(rng, Vector(1:40))
for i in 1:10
    ps[i]=temp[i]
end
colors= Array{RGB{FixedPointNumbers.Normed{UInt8,8}}}(undef,0)
po=0
for i in 1:40
    po=0
    for j in 1:10
        if (ps[j]==i)

            push!(colors,colorant"#389826")
            po=1
            break
        end
    end
    if(po==0)
        push!(colors,colorant"#CB3C33")
    end




 end
println("The patrol centers are on the node:",ps)



finalCitymap=zeros(40,40)
for i in 1:40
    for j in 1:40
        finalCitymap[i,j]=rawmap[i,j]
    end
end



plt=graphplot(rawmap,

          nodeshape=:circle,
```

```
          nodesize=0.1,

          curves=false,
          names = 1:40,
          fontsize = 10,
            nodecolor=colors,
          linecolor = :darkgrey
          )


n=40

for i in 1:n
    for j in 1:n
        if (i==j)
            rawmap[i,i]=0

        elseif ( rawmap[i,j]==0)
            rawmap[i,j]=10000
        end
    end
end
#println(rawmap)

for k in 1:n
    for i in 1:n
       for j in 1:n
          if(rawmap[i,j]>rawmap[i,k]+rawmap[k,j])
                rawmap[i,j]=rawmap[i,k]+rawmap[k,j]
            end

        end
    end
end

#println(rawmap)
covermap=zeros(10,40)
bcovermap=zeros(10,40)
for i in 1:10
       for j in 1:40
          if(rawmap[Int(ps[i]),j]<3)
               covermap[i,j]= rawmap[Int(ps[i]),j]
               bcovermap[i,j]=1
            end

        end
end
#println(covermap)
#println(bcovermap)
for i in 1:10

    for j in 1:40
        if(j in ps)
           bcovermap[i,j]=0
        end
    end
end
```

```
printcovermap=zeros(40,40)

for i in 1:10
    for j in 1:40
          if(covermap[i,j]>0)
             printcovermap[i,j]=covermap[i,j]
           end
    end
end

######################################################################
#####################################################################
                   #solution of the problem
######################################################################
#####################################################################


function find(i,mat,usd,m,n,match) #function of hugarian algorithmn
        #println("i:",i,"j:",j)
       for j in 1:n
               # println("i:",i,"j:",j)
              if (mat[Int(i),j]==1&&usd[j]!=1)

                    usd[j]=1
                    if(match[j]==0||(find(Int(match[j]),mat,usd,m,n,match)))
                       match[j]=i
                       return true
                     end
               end

       end


    return false
end

cmt=temp[11]
vcar=0.1
vpcar=0.16
#there are t=100 limit
t=0
######################################################################
#####################################################################
                   #main loop to find min(t)
######################################################################
#####################################################################
while(t<100)
   t+=1
    rc=0
   rc=t*vcar+0.3
    rp=t*vpcar
  setB=[]
    println("Iterating t=",t)
   # println("rc",rc)
   for i in 1:40
        if(rawmap[Int(cmt),i]<rc)
            push!(setB,i)
        end
   end
    maA=zeros(10,length(setB))
    start=zeros(2)
```

```
    for i in 1:10
        for j in 1:length(setB)
           if(rawmap[Int(ps[i]),setB[j]]<rp)
                #println("d():",rawmap[Int(ps[i]),setB[j]]," < ",rp)
                maA[i,j]=1
               start[1]=i
                start[2]=j
           end
        end
    end
    println("At this time criminal's range: ",setB," starting from node :" ,temp[11
])
    println("The roads that patrol center can block:",maA)
    #println(start)
    #if(length(setB)==0||start[1]==0)
        #continue
    #end
    flg=0
     match=zeros(length(setB))
    for i in 1:10
       used=zeros(length(setB))

       if( find(i,maA,used,10,length(setB),match)==true)
            perfect =1
            for i in 1:length(setB)
                if (match[i]==0)
                    perfect=0
                end
            end
            if (perfect==1)
              println("Match found",match," at time t=",t)
              flg=1
               break
            end
        end
         println("current Match",match)
    end

    if(flg==1)

        break
    end

end
```

```
The patrol centers are on the node:[28.0, 38.0, 31.0, 32.0, 33.0, 18.0,
37.0, 9.0, 27.0, 14.0]
Iterating t=1
At this time criminal's range: Any[39] starting from node :39
The roads that patrol center can block:[0.0; 0.0; 0.0; 0.0; 0.0; 0.0; 0.
0; 0.0; 0.0; 0.0]
current Match[0.0]
current Match[0.0]
current Match[0.0]
current Match[0.0]
current Match[0.0]
current Match[0.0]
current Match[0.0]
current Match[0.0]
current Match[0.0]
current Match[0.0]
Iterating t=2
At this time criminal's range: Any[39] starting from node :39
The roads that patrol center can block:[0.0; 0.0; 0.0; 0.0; 0.0; 0.0; 0.
0; 0.0; 0.0; 0.0]
current Match[0.0]
current Match[0.0]
current Match[0.0]
current Match[0.0]
current Match[0.0]
current Match[0.0]
current Match[0.0]
current Match[0.0]
current Match[0.0]
current Match[0.0]
Iterating t=3
At this time criminal's range: Any[36, 39] starting from node :39
The roads that patrol center can block:[0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0
0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
Iterating t=4
At this time criminal's range: Any[36, 39] starting from node :39
The roads that patrol center can block:[0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0
0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
Iterating t=5
At this time criminal's range: Any[36, 39] starting from node :39
The roads that patrol center can block:[0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0
```

```
0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
Iterating t=6
At this time criminal's range: Any[36, 39] starting from node :39
The roads that patrol center can block:[0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0
0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
Iterating t=7
At this time criminal's range: Any[36, 39] starting from node :39
The roads that patrol center can block:[0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0
0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
Iterating t=8
At this time criminal's range: Any[36, 39] starting from node :39
The roads that patrol center can block:[0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0
0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
Iterating t=9
At this time criminal's range: Any[36, 39] starting from node :39
The roads that patrol center can block:[0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0
0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0; 0.0 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
```

```
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
current Match[0.0, 0.0]
Iterating t=10
At this time criminal's range: Any[1, 36, 39] starting from node :39
The roads that patrol center can block:[0.0 0.0 0.0; 0.0 0.0 0.0; 0.0 0.
0 0.0; 0.0 0.0 0.0; 0.0 0.0 0.0; 0.0 0.0 0.0; 0.0 0.0 0.0; 0.0 0.0 0.0;
0.0 0.0 0.0; 0.0 0.0 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
Iterating t=11
At this time criminal's range: Any[1, 36, 39] starting from node :39
The roads that patrol center can block:[0.0 0.0 0.0; 0.0 0.0 0.0; 0.0 0.
0 0.0; 0.0 0.0 0.0; 0.0 0.0 0.0; 0.0 0.0 0.0; 0.0 0.0 0.0; 0.0 0.0 0.0;
0.0 0.0 0.0; 0.0 0.0 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
Iterating t=12
At this time criminal's range: Any[1, 36, 39] starting from node :39
The roads that patrol center can block:[0.0 0.0 0.0; 0.0 0.0 0.0; 0.0 0.
0 0.0; 0.0 0.0 0.0; 0.0 0.0 0.0; 0.0 0.0 0.0; 0.0 0.0 0.0; 0.0 0.0 0.0;
0.0 0.0 0.0; 1.0 0.0 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[10.0, 0.0, 0.0]
Iterating t=13
At this time criminal's range: Any[1, 36, 39] starting from node :39
The roads that patrol center can block:[0.0 0.0 0.0; 0.0 0.0 0.0; 0.0 0.
0 0.0; 0.0 0.0 0.0; 0.0 0.0 0.0; 0.0 0.0 0.0; 0.0 0.0 0.0; 0.0 0.0 0.0;
0.0 1.0 0.0; 1.0 0.0 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
```

```
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0]
current Match[0.0, 9.0, 0.0]
current Match[10.0, 9.0, 0.0]
Iterating t=14
At this time criminal's range: Any[1, 30, 36, 39] starting from node :39
The roads that patrol center can block:[0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.
0; 0.0 0.0 1.0 0.0; 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0;
0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0; 0.0 0.0 1.0 0.0; 1.0 1.0 0.0 0.0]
current Match[0.0, 0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0, 0.0]
current Match[0.0, 0.0, 3.0, 0.0]
current Match[0.0, 0.0, 3.0, 0.0]
current Match[0.0, 0.0, 3.0, 0.0]
current Match[0.0, 0.0, 3.0, 0.0]
current Match[0.0, 0.0, 3.0, 0.0]
current Match[0.0, 0.0, 3.0, 0.0]
current Match[0.0, 0.0, 3.0, 0.0]
current Match[10.0, 0.0, 3.0, 0.0]
Iterating t=15
At this time criminal's range: Any[1, 30, 36, 39] starting from node :39
The roads that patrol center can block:[0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.
0; 0.0 0.0 1.0 0.0; 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0; 1.0 1.0 0.0 0.0;
0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0; 0.0 0.0 1.0 0.0; 1.0 1.0 0.0 0.0]
current Match[0.0, 0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0, 0.0]
current Match[0.0, 0.0, 3.0, 0.0]
current Match[0.0, 0.0, 3.0, 0.0]
current Match[0.0, 0.0, 3.0, 0.0]
current Match[6.0, 0.0, 3.0, 0.0]
current Match[6.0, 0.0, 3.0, 0.0]
current Match[6.0, 0.0, 3.0, 0.0]
current Match[6.0, 0.0, 3.0, 0.0]
current Match[10.0, 6.0, 3.0, 0.0]
Iterating t=16
At this time criminal's range: Any[1, 29, 30, 36, 39] starting from node
:39
The roads that patrol center can block:[0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0
0.0 0.0; 0.0 0.0 0.0 1.0 0.0; 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 1.0 0.0 0.0;
1.0 0.0 1.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0; 0.0 1.0 0.0 0.0 0.0; 0.0 0.0
0.0 1.0 1.0; 1.0 0.0 1.0 0.0 0.0]
current Match[0.0, 0.0, 0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0, 0.0, 0.0]
current Match[0.0, 0.0, 0.0, 3.0, 0.0]
current Match[0.0, 0.0, 0.0, 3.0, 0.0]
current Match[0.0, 0.0, 5.0, 3.0, 0.0]
current Match[6.0, 0.0, 5.0, 3.0, 0.0]
current Match[6.0, 0.0, 5.0, 3.0, 0.0]
current Match[6.0, 8.0, 5.0, 3.0, 0.0]
Match found[6.0, 8.0, 5.0, 3.0, 9.0] at time t=16
```

Result and Interpretation

It seems like if the criminal is last seen in this random traffic road, police car needs to be 1.5 times faster than the car of criminal in order to catch him. Because the selection of patrol centers is random and the result won't be too good .

Obviously not every traffic road can be intercepted, because although the siren did work you can't drive 110 mph in the city. Other fun things to do would be pack the main of the program above and pass combinations of police station in to find a better maximum possible intercep node of the whole city.

Conclusion

To summarize this project, the original three questions are almost solved, although there are some limitations to fit in the realistic situation. (e.g. Recall the Integer Program in Problem1) One of my findings is that for a city (given the coordinates of every traffic road and the coordinates of every patrol center), it is possible to use warshall alogrithmn to find out the distance between any two patrol centers and traffic roads, as well as which traffic roads will be covered in each patrol center. Finally we can get a optimized solution (valued by the evenness of police cases accepted by all centers) on how to distribute the police force to each traffic road. Another finding is that for problem3, we find the ratio of the speed of the police and the speed of the crimnal such that the criminal can be taken down needs to be 1.5 times.

There are many aspects we can investigate in the future, we can implement another obective function and plot the tradeoff curve between the the balance of the patrol distribution and the efficient of patrol distribution.

It would be also interesting to discuss if there is a correlation betweem the second objective in the first problem and their performance in second problem. Though first we would need to figure out how to evaluate their performance in problem 2. But this can be evaluated by suming all the objective by calculating the linear combinations (mutiply by weight).

In [ ]:

```

```