Report 4: Circle Packing and Maze Setting

Wu Lan

2019/8/8

1 Introduction

This report focuses on a more complex setting of mazes in order to prove DCCP's ability in shortest path planning under different scenarios. So firstly, the circle packing algorithm is implemented still with the DCCP package, showing a list of circles externally tangent with each other packing in the smallest square. Therefore, we can get a suitable pattern setting avoiding possibility of overlap. Similarly, we can also change the element of maze into square or obtain a mix of square and circle seeing which pattern looks like the original obstacle to a larger extent. To set the overall pattern more likely to be a maze, I also adjust parameters and leave gap among circles. After that, we randomly choose several starting points and ending points to do the route-planning. The results will be shown in the following report. What's more, some of my simulations of the Pembroke map will be demonstrated in the end.

2 Method and Result

- Circle Packing
  In order to package circles inside a minimum square, we need to set the place of their origin one by one. For example, assuming that there are totally m circles (c_1, c_2,$\cdots c_m$) with corresponding radius (namely r_i) and origin (x_i and y_i), we need to check the place from the first to the end. For circle c_j, it should be compared with circles whose number larger than i.

$$\sqrt{(x_j - x_q)^2 + (y_j - y_q)^2} \preceq r_j + r_q \tag{1}$$

  where q > j.
  Finally we minimize the area circles occupy with the following code:

```
constr = []
for i in range(n-1):
for j in range(i+1, n):
constr.append(cvx.norm(cvx.vec(c[i,:]-c[j,:]), 2) >= r[i]+r[j]+0.1)
prob1 = cvx.Problem(cvx.Minimize(cvx.max(cvx.max(cvx.abs(c), axis=1) + r)
                                         ), constr)
prob1.solve(method = 'dccp', solver='ECOS', ep = 1e-2, max_slack = 1e-2)
```

Here are figures of maze setting created by the circle packing algorithm:

```
#Parameter Setting of Figure 1
r = np.linspace(1,4,10)
#Parameter Setting of Figure 2
r = 1
#Parameter Setting of Figure 3
r = np.linspace(1,2,50)
```

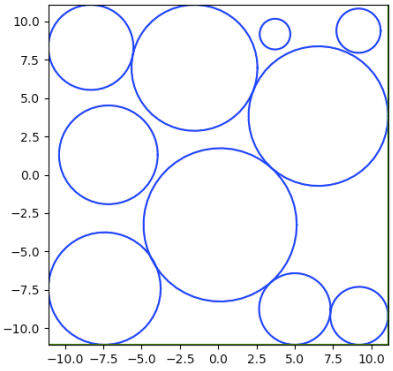

Figure 1: n = 10

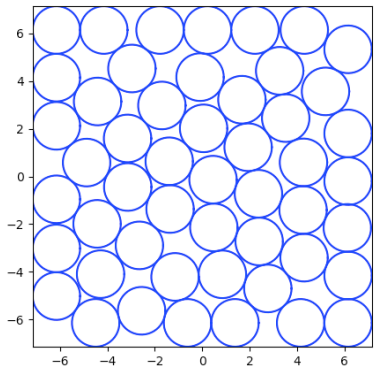

Figure 2: n = 50

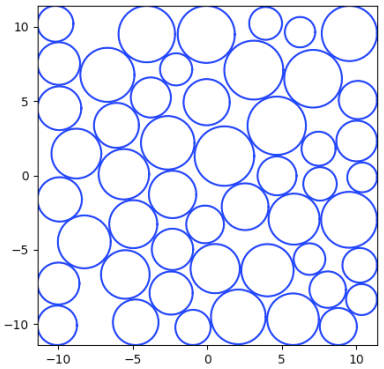

Figure 3: n = 50

In order to make those figures more like a maze with gap, the equation changes to

$$\sqrt{(x_j - x_q)^2 + (y_j - y_q)^2} \preceq r_j + r_q + 0.1 \tag{2}$$

and the result is as following with specific starting point and ending point.

```
x = []
for i in range(50+1):
x += [cvx.Variable((2, 1))]
L = cvx.Variable(1)
constr = [x[0] == a, x[50] == b]
cost = L
for i in range(50):
constr += [cvx.norm(x[i]-x[i+1]) <= L/50]
for j in range(50):
constr += [cvx.norm(x[i]-p[:,j]) >= r[j]]
prob2 = cvx.Problem(cvx.Minimize(cost), constr)
print "begin to solve"
result = prob2.solve(method='dccp')
print "end"
```

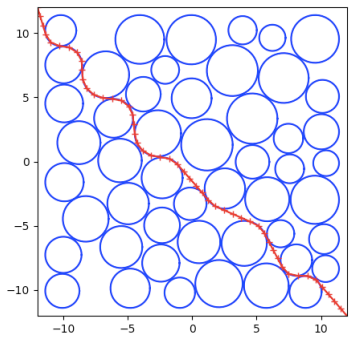

Figure 4: S: (-12, 12) E: (12, -12)

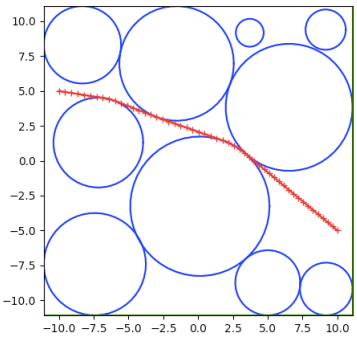

Figure 5: S: (-10, 5) E: (10, -5)

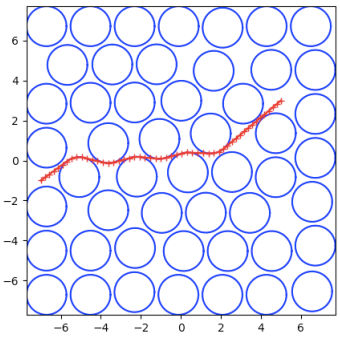

Figure 6: S: (-6.5, -1) E: (5, 3)

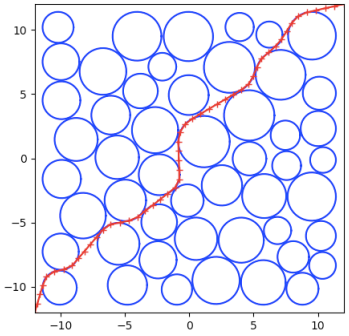

Figure 7: S: (-12, -12) E: (12, 12)

- Square Packing

  Similarly, we use square to replace circles in the code above following the same origin set method. Still by minimizing the total area, we get settings:

```
constr = [x[0] == a, x[n] == b]
cost = L
for i in range(n):
constr += [norm(x[i]-x[i+1]) <= L/n]
for j in range(50):
constr += [abs(x[i][0]-p[:,j][0]) + abs(x[i][1]-p[:,j][1]) >= r[j]]
prob = Problem(Minimize(cost), constr)
result = prob.solve(method='dccp')
```

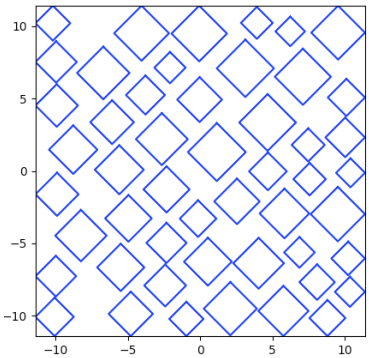

Figure 8: Setting

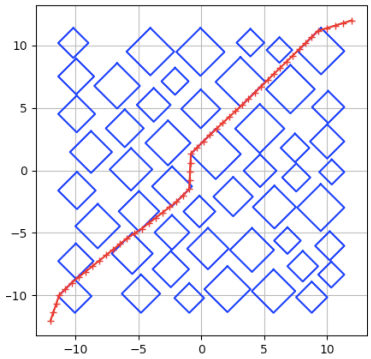

Figure 9: S: (-12, -12) E: (12, 12)

3 Basic Map Simulation

Here is a map of the Pembroke College, and I will take a part of it to do the path planning.

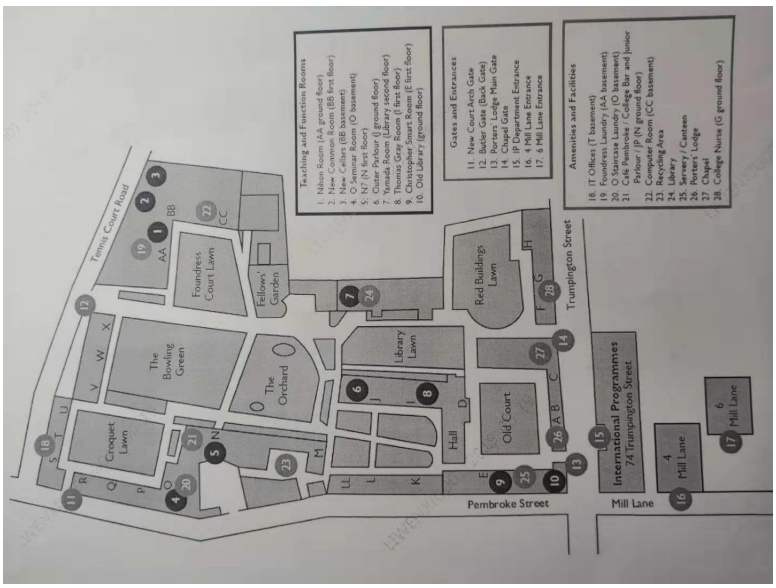

Figure 10: Pembroke College Map

Each of the building will be represented by a list of circles in order to simulate the shape of it. After that, the DCCP package will be applied to find the shortest path.

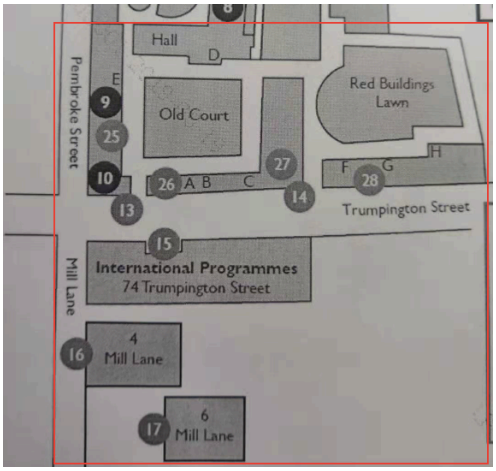

Figure 11: Simulation Part of the Map

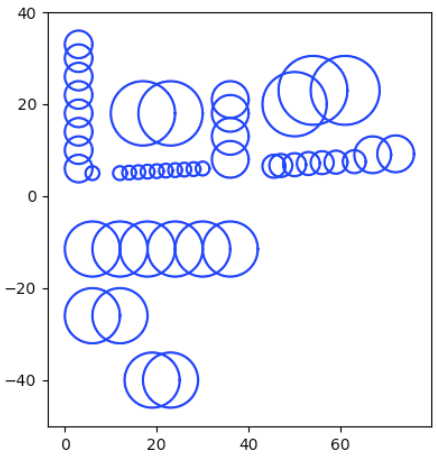

Figure 12: Simulation

Errors appear like what shown in the following picture. On one hand, the number of point we set in order to plan the route is small (here $n = 50$). When two points are far away with each other, they may turn out to be a line passing the circle of building. On the other hand, the error also indicates that the computation process is quite complex, and thus not suitable to be applied to the reality.

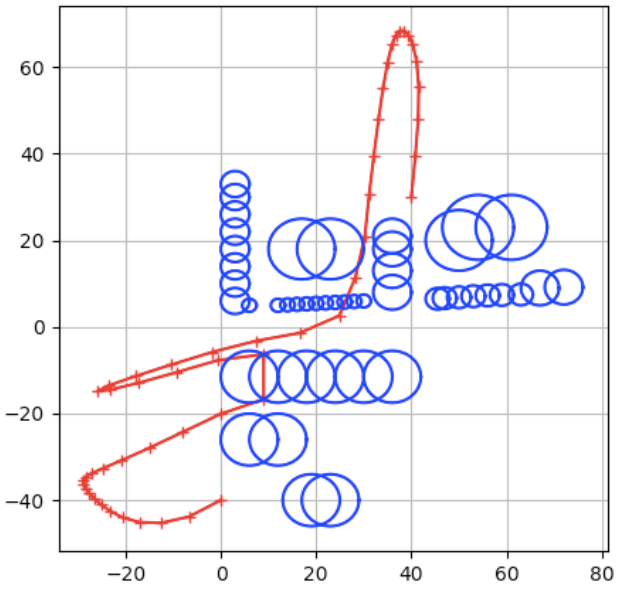

Figure 13: Route Planning with error