Report 3: DCCP Simulation in Different Scenarios

Wu Lan

2019/8/2

1 Introduction

This report will briefly introduce the different outcomes as well as simulation settings with DCCP implementation. On one hand, I recognize obstacles and turn them into different shapes like circle and rectangle in order to implement the shortest path algorithm. On the other hand, I also discuss different scenarios if we randomly set the starting point and determination or change the properties of obstacles in terms of scale and location. In addtion, at the end of the report, I provide part of my understanding about 3D route planning in technical perspective.

2 Method

In order to turn obstacles into different shapes, I simply take two distance calculation method when setting the constraints in DCCP program. As shown in the following figures, the Euclidian distance in 2-dimension space will naturally form a circle while the Manhattan distance can form a square, which largely influence the route-planning outcome.

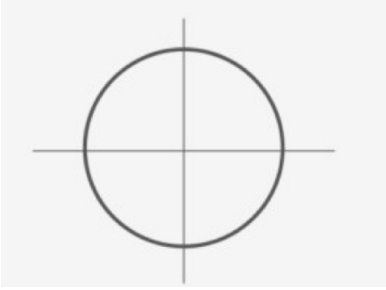


Figure 1: Euclidean Distance (d_1)

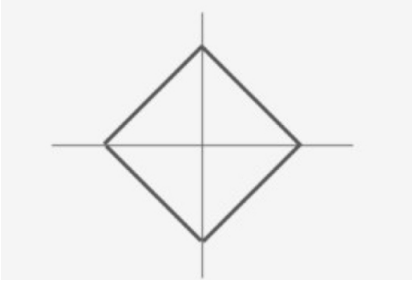


Figure 2: Manhattan Distance (d_2)

$$d_1(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2} \tag{1}$$

$$d_2(I_1, I_2) = \sum_p |I_1^p - I_2^p| \tag{2}$$

- Constraints of Circle Obstacles:

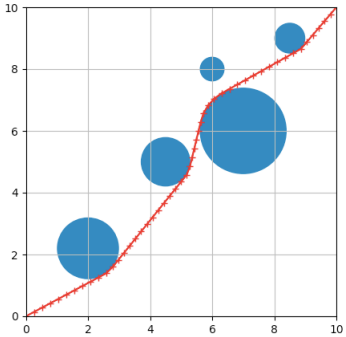
```
x = []
for i in range(n+1):
x += [Variable((d, 1))]
L = Variable(1)
constr = [x[0] == a, x[n] == b]
cost = L
for i in range(n):
constr += [norm(x[i]-x[i+1]) <= L/n]
for j in range(m):
constr += [norm(x[i]-p[:,j]) >= r[j]]
prob = Problem(Minimize(cost), constr)
```

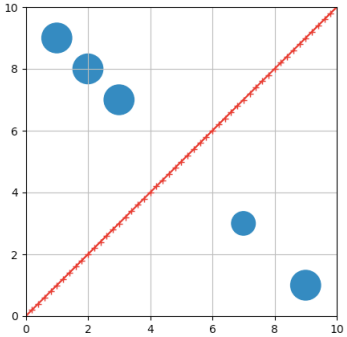
- Constraints of Rectangular Obstacles:

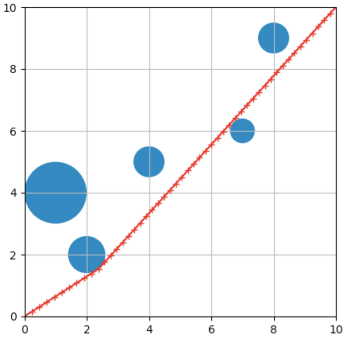
```
x = []
for i in range(n+1):
x += [Variable((d, 1))]
L = Variable(1)
constr = [x[0] == a, x[n] == b]
cost = L
for i in range(n):
constr += [norm(x[i]-x[i+1]) <= L/n]
for j in range(m):
constr += [abs(x[i][0]-p[:,j][0]) + abs(x[i][1]-p[:,j][1]) >= r[j]]
prob = Problem(Minimize(cost), constr)
print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

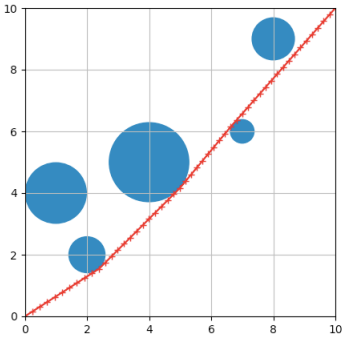
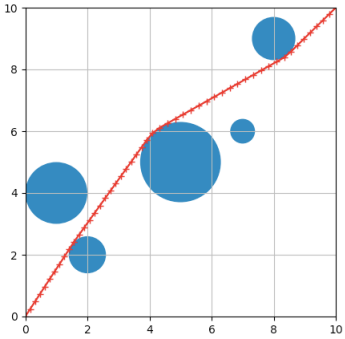
3 Result

3.1 Circle & Settled Starting/Ending Points



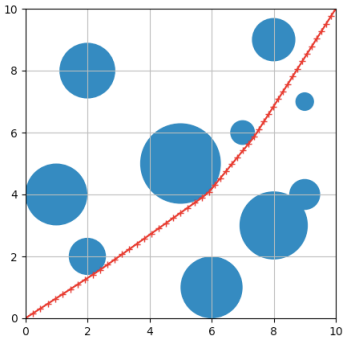
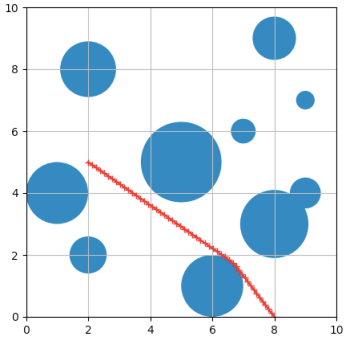
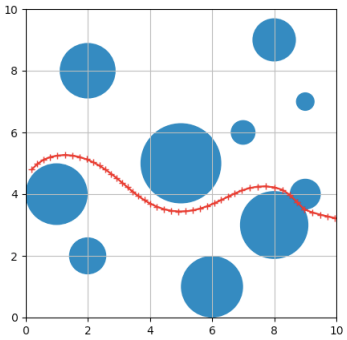
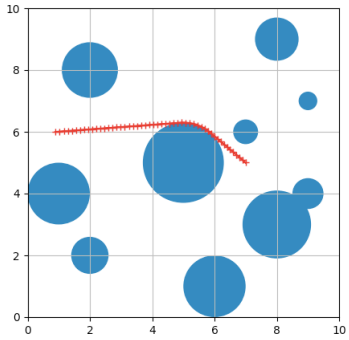
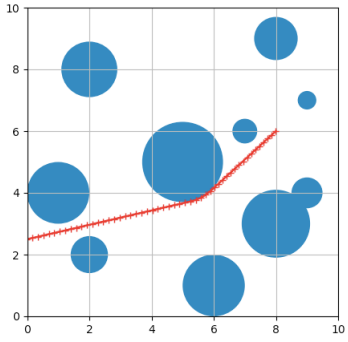




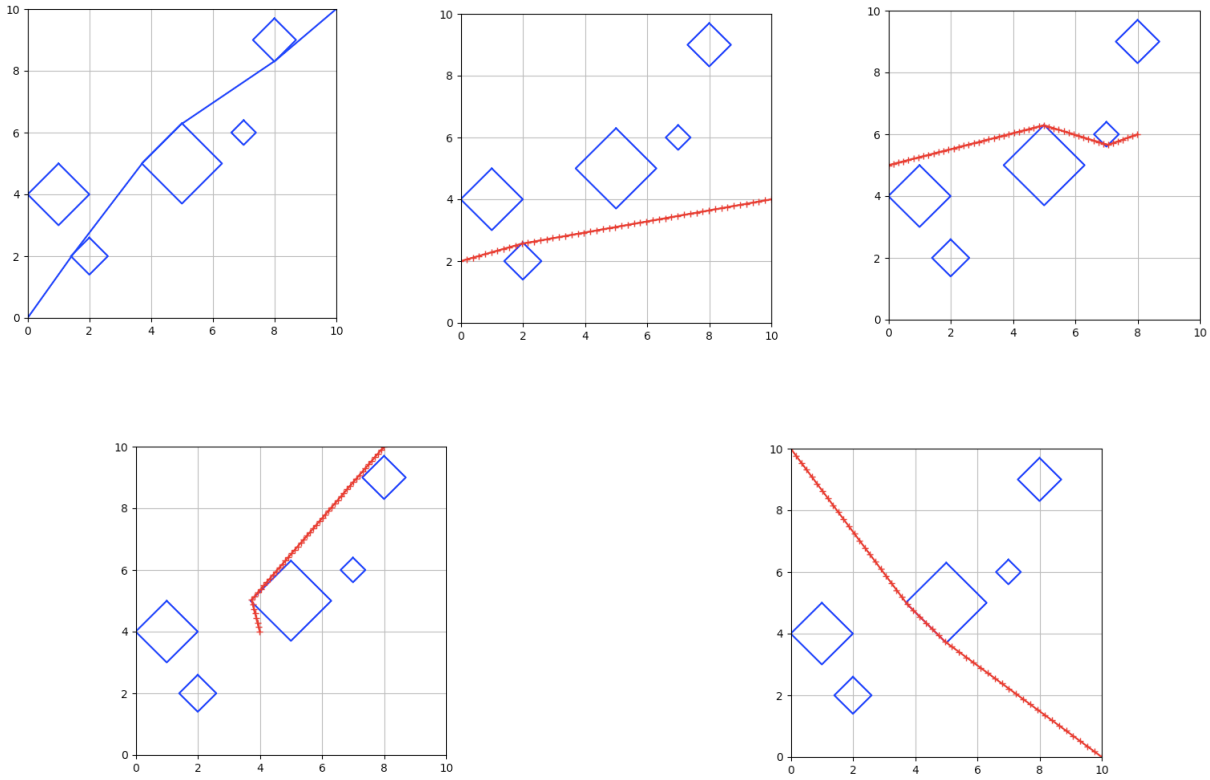
With settled starting point and ending point, we can find that the shape of the planned route is influenced by the location of the obstacles. In some cases, if the obstacles are placed away from the line connected two points, the shortest path is absolutely the straight line. And when the original path is influenced by obstacles to a small extent, the planned route appear to be conterminous segments tangent to those circles in the way. Also, in the cases that obstacles impact the previous straight line to a large extent, curves will be the main element of the final shortest path. Apparently, those curves are part of the periphery of obstacles in most situations.

3.2 Circle & Random Starting/Ending Points

Outcomes in these figures are quite similar to settled S/E situations. However, one point should be highlighted is that in the third figure above the route goes through the narrow gap between two circles. On one hand, in real circumstances, we should avoid this kinds of planned route for safety concerns. On the other hand, given that car also has volume, we should better set the radius of circle much larger.

3.3 Rectangle & Random Starting/Ending Points



For rectangle obstacles, the shape of route will be much easier since it consists of segments only. It also can be a straight line, but most time it appears to be conterminous segments containing the vertex and side of the rectangle.

4 Some Implications

4.1 Smoth Path Planning

For the purpose of obtaining a smooth and secure route, the shortest path may not be a good choice. Therefore we can introduce another variable u representing the direction vector of each point: $u_i = x_(i+1) - x_i$. We can minimize the sum of square of the u_i matrix and finally get the outcome.

```
x = Variable((d,n+1))
u = Variable((d,n+1))
constr = [x[:,0][0] == a, x[:,n][0] == b, u[:,0][0] == c]
cost = sum_squares(u[:, 1:] - u[:, :-1])
constr += [x[:,1:] == x[:,:-1] + dt*u[:, :-1]]
for i in range(n):
for j in range(m):
constr += [norm(x[:,i][0]-p[:,j]) >= r[j]]
constr += [x[:,1:] == x[:,:-1] + dt*u[:, :-1]]
prob = Problem(Minimize(cost), constr)
```

4.2 3D Route Planning

- Aerial Route Planning

```
x = []
for i in range(n+1):
x += [Variable((3, 1))]
L = Variable(1)
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(m):
constr += [norm(x[i]-p[:,j]) >= r[j]]
prob = Problem(Minimize(cost), constr)
```

For Aerial planning, it will be quite similar to 2D planning since we just turn circle into sphere and DCCP program can also provide calculation function. Simply, we turn d into 3 and code in the same style.

- Surface Route Planning