Parking Conundrum

Karthik Anantharaman
Department of Aeronautics
Astronautics
Stanford University
ka99@stanford.edu

Abstract

The number of cars have increased manifold in the recent years, and with electric vehicles bearing lower running costs promised for the future, the numbers don't seem to be decreasing. While the number of cars increase, the land available remains constant and hence the amount of space available to park these cars. While multistoried car parks are an option, a large amount is spent in the infrastructure and hence high charges are levied on parking to recover the expenses. Maximum utilization of the existing space by tweaking the layout seems like an excellent solution with limited finances. This project aims to maximize the parking space available to accommodate maximum number of parking spots while also maintaining sufficient road network within the plot. The aim is to optimize the parking designs for different shapes of available parking spaces.

1 Introduction

Its funny how parking lots were once built to cater to restaurants and eateries, but now restaurants are built around parking lots. Many parking lots have eateries, washrooms and facilities for those who park their car but do they really have space for the purpose they came for - park.

1.1 Requirements of an ideal parking space

1. Maximum parking spots.
2. Easy ingress and egress to the spot.
3. Ordered road network.

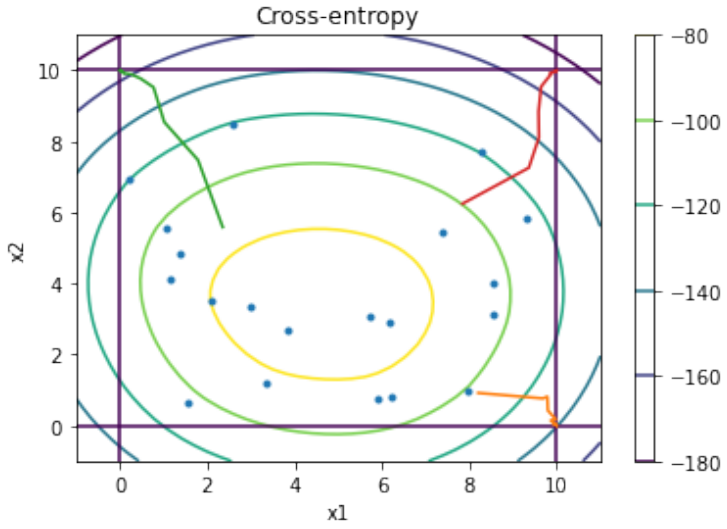
2 Approach

Broke down the overall problem into separate objectives and tackled them separately. For a specific piece of area, maximized the number of parking spots that can be assigned in the region. Generated an optimal road network based on both the area as well as the positioning of parking spots.

3 Parking spots

In a certain square subdomain aimed to maximize the number of parking spots that could be accommodated in the region.

Started off with treating each parking spot as a point in the domain. With an initial distribution of points already in the area, added a new point such that it is farthest away from each of the initial point. The boundary were the only constraints. The objective function therefore was to maximize the sum of distances to the new point from each of the points already in the domain. It was equivalent to minimizing the inverse of the distance.

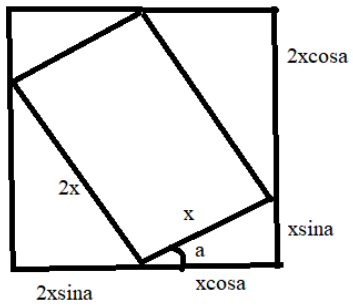


Used the cross entropy method for optimizing the algorithm and observed that it placed a new point at the corner of the boundary which is expected since it is farthest from the existing points in the domain.

It could be observed that as points kept getting added, a chessboard pattern appeared. This was expected as the diagonal distance ensured maximum distance between points. This design would however not work as vehicles would not have access to the parking spot as a road would not be able to connect each of these points.

For a one dimensional domain the uniform configuration produced the least cost for the objective function and was hence the optimal. Thus suggesting the equally spaced vehicles in a line were the ideal setup.

It was now therefore a task to determine at which angle the cars should be placed in a line. Treating each parking spot as a rectangle of ratio 2:1 suggested that placing it either horizontally or vertically took the least area.



The area in the center is the parking spot $(2x * x)$ and the outer area is the area occupied by this parking spot $((2xsina + xcosx) * (2xcosa + xsina))$. Since a varies from 0 to 90 degree, therefore the area is minimized for a equal to 0 or 90 degree.

While placing it vertically requires us to leave some space for the doors to open, placing them horizontally would require to leave a much more significant area, sufficient to reverse and squeeze the car into. Therefore, vertically placed rectangles, stacked one next to each other in a line was the optimal parking space strategy for any subdomain. Two rows of cars could be placed facing each other, with a connecting road on either side.

3.1 Code

See Appendix A.

4 Road

Simplified the problem to consist of a closed loop road network within the parking space. A closed loop road ensures that the vehicles can follow the path in a clockwise or anticlockwise direction (depending on whether it is a left hand or right hand drive country) and hence a single lane road would suffice. This saves on area of a double lane road thus increasing the area available for parking. A single connection to the entry/exit points can also be drawn from any point in the loop and would thus ensure minimum hassle with no unnecessary branches in the road system. Hence, the simplification is justified.

4.1 Domain and parameters

- Worked on a unit square domain, however the logic can be generalized.
- Chose random 100 points within the domain that indicated the area of the domain.
- Started with a random closed path consisting of a certain number of points (20) which acted as the initial road network.
- A parameter characterizing the width of the road.

4.2 Constraints

Constraint were designed to ensure:

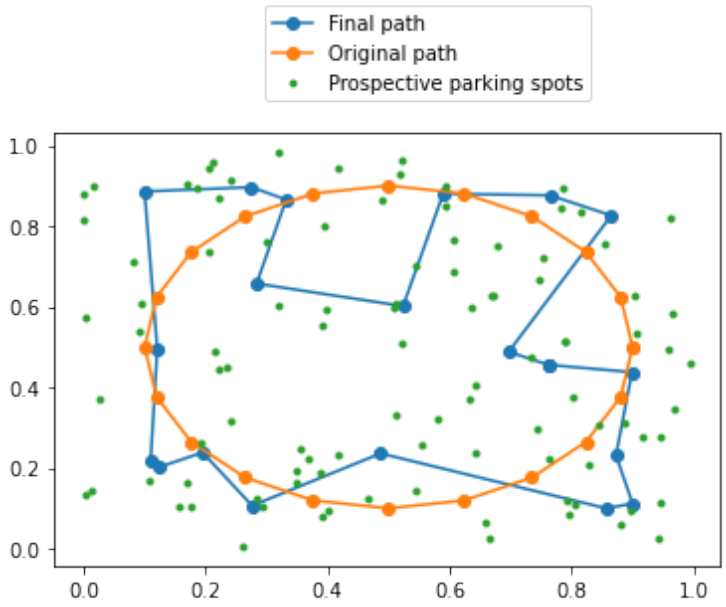
- The road is not close to the boundary.
- Not more than one segment of the road are near a single point in the domain.
- All points in the domain are close to atleast one segment of the road.
- The road does not comprise of sharp bends.

4.3 Optimization function

Looped over all points in the domain and associated a penalty based on constraint violations which added up to form the cost. The objective was to minimize the cost and the configuration with the minimum cost would be the optimal closed road path.

4.4 Observations

Started off with a random initial closed loop configuration (a circle) and used the Nelder-Mead method to optimize the path.



4.5 Discussion of Results

It can we observed that the initial path is a really poor design. While it may be accessible to points that are close to the boundary, a huge void is present in the center. It therefore means, parking spots towards the center would have to travel long distances to reach the road. The path generated after iterating over the algorithm performs much better and tries to maintain a path that is equidistant from all the points in the domain. The same can be observed from the costs.

Path	Cost
Initial	34.924
Final	3.38

Table 1: Optimization function value for the initial and final path configurations

4.6 Code

See Appendix B.

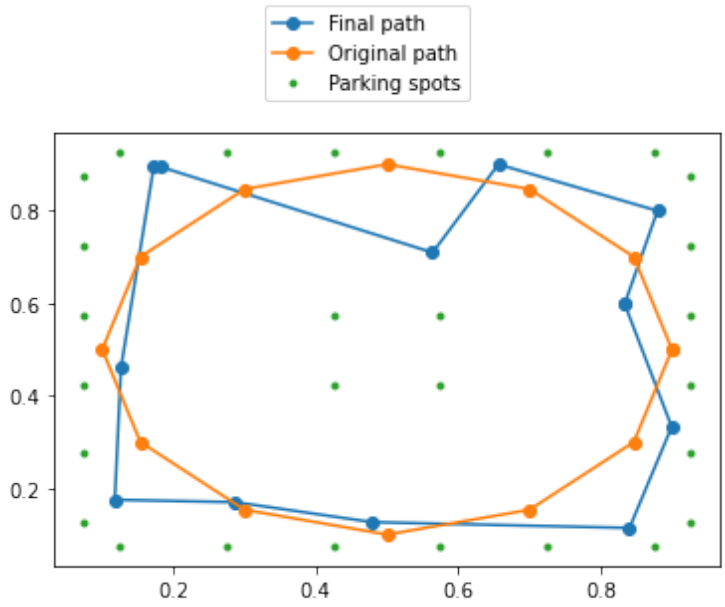
5 Combining the two

We now know a strategy that maximizes parking spaces and creates paths for a domain. We need to now generate a path for a design we know maximizes parking spaces. Lets look at an example for which we sort of the know what the outcome should be.

A parking lot with parking spots placed along the perimeter along with two rows in the center, should have a rectangular road surrounding the center block that can be accessed by vehicles both in the center as well as those along the perimeter.

Started off with a random initial path (a circle) and used the Nelder-Mead method to try and optimize the path. The domain, parameters, constraints and optimization function are identical to the previous problem except that the parking spots are specified.

5.1 Observations and Discussion of Results



We can see that the path tries to manifest itself into a rectangle that is not only equidistant from the points in the center but also from those along the boundary. It compares with the ideal path expected for this configuration and hence justifies the working of this algorithm. The algorithm thus managed to generate an optimal path for a region while maximizing the number of parking spots, producing the best parking lot design possible,

5.2 Code

See Appendix C.

6 Related Work

Significant references related to the problem were not available for the problem. Porter et al. [2013] was the only paper that had something related to what I was working on and was also a great validation checkpoint to compare some findings.

Benefited immensely the textbook Algorithms for Optimization for implementation of all the techniques and strategies in the project.

7 Conclusion

The project created a simple implementation of attempting to solve the parking crisis. It managed to maximize parking spaces and keep them well connected with a good road network. While a little more polished design is needed for final implementation, the outputs generated were close to what is expected and gives an idea of what an ideal parking lot should look like. Though I prefer cost efficient and hassle free public transport, I hope if you ever have to use your car you find a parking spot, and if you don't, do remember this project.

References

Richard J. Porter, John Billingham, Joel Bradshaw, Marcin Bulkowski, Peter Dawson, Pawel Garbacz, Mark Gilbert, Lara Goscé, Poul Hjort, Martin E. Homer, Mike R. Jeffrey, Dario Papavassiliou, and David Kofoed Wind. Optimisation of car park designs. 2013.

Appendix A: Parking Space Maximization

```
def cross_entropy(f, c, x, x0, rho, gamma, nw, nel, nf, var, cov_fac):
    xhist = [x0]
    fhist = [f(x,x0)]
    chist = [max(max(c(x0)[0],0),max(c(x0)[1],0))]
    mean = x0
    cov = var*np.identity(len(x0))
    for k in range(20):
        rho *= gamma
        s = np.random.multivariate_normal(mean,cov,nw)
        vals = np.array([f(x,s[i]) for i in range(nw)])
        cs = np.array([c(s[i]) for i in range(nw)])
        val1 = np.zeros_like(vals)
        for i in range(len(cs)):
            val1[i] = vals[i]+rho*np.sum(np.maximum(cs[i],np.zeros_like(cs[i]))**2)
        elite = val1.argsort()[:nel]
        el_vals = np.array([s[i] for i in elite])
        mean = np.mean(el_vals, axis=0)
        cov *= cov_fac
        xhist.append(mean)
        fhist.append(f(x, mean))
        chist.append(max(max(c(mean)[0],0),max(c(mean)[1],0)))
        nw = int(nw*nf)
    return xhist, fhist, chist

def f1(x,x1):
    s = 0
    for i in range(len(x)):
        s+=np.linalg.norm(x[i]-x1)
    return -s

def f3(x,x1):
    s = 0
    for i in range(len(x)):
        s+=1/np.linalg.norm(x[i]-x1)
    return s

def c1(x1):
    return [-x1[0],-x1[1],x1[0]-10,x1[1]-10]

def each_iter(x):
    xhist1, fhist1, chist1 = cross_entropy(f3, c1, x, 10*np.random.random(2), 100, 3, 50, 5, 1, 2
    x = list(x)
    x.append(xhist1[-1])
    x = np.array(x)
    return x

xhist1, fhist1, chist1 = cross_entropy(f1, c1, x, 10*np.random.random(2), 100, 3, 50, 5, 1, 2, 0.
xhist2, fhist2, chist2 = cross_entropy(f1, c1, x, 10*np.random.random(2), 100, 3, 50, 5, 1, 2, 0.
xhist3, fhist3, chist3 = cross_entropy(f1, c1, x, 10*np.random.random(2), 100, 3, 50, 5, 1, 2, 0.

xr = np.linspace(-1,11,41)
yr = np.linspace(-1,11,41)

X,Y = np.meshgrid(xr,yr)

F = np.zeros_like(X)
for i in range(len(X)):
```

```
    for j in range(len(Y)):
        F[i,j] = f1(x,[X[i,j],Y[i,j]])
plt.contour(xr,yr,F,levels=[-1000000,-160,-140,-120,-100,-80,10000])
# plt.contour(xr,yr,F,levels=[80,100,120,140,160,180])
plt.plot(x[:,0],x[:,1],'.')
plt.plot([xhist1[i][0] for i in range(len(xhist1))], [xhist1[i][1] for i in range(len(xhist1))])
plt.plot([xhist2[i][0] for i in range(len(xhist2))], [xhist2[i][1] for i in range(len(xhist2))])
plt.plot([xhist3[i][0] for i in range(len(xhist3))], [xhist3[i][1] for i in range(len(xhist3))])
plt.xlabel('x1')
plt.ylabel('x2')
# plt.clim(0,10)
# plt.colorbar()
plt.contour(xr,yr,np.array(c1([X,Y]))[0],levels=[0])
plt.contour(xr,yr,np.array(c1([X,Y]))[1],levels=[0])
plt.contour(xr,yr,np.array(c1([X,Y]))[2],levels=[0])
plt.contour(xr,yr,np.array(c1([X,Y]))[3],levels=[0])
plt.title('Cross-entropy')
plt.savefig('s1o3.png')
plt.show()

x = np.array([np.zeros(2)])
for i in range(100):
    x = each_iter(x)

plt.plot(x[:,0],x[:,1],'.')
```

Appendix B: Parking Space Maximization

```
def f(xb,Nb,xd,Nd,w):

    xb_closed = list(xb.copy()[:2*Nb])
    xb_closed.append(xb[0])
    xb_closed.append(xb[1])
    xb_closed = np.array(xb_closed)
    xb_closed = np.reshape(xb_closed,(Nb+1,2))

    cost = 0;
    #Distance from boundary
    for i in range(Nb):
        if xb_closed[i,0]<w:
            cost = cost+1e20*(w-xb_closed[i,0])
        if xb_closed[i,1]<w:
            cost = cost+1e20*(w-xb_closed[i,1])
        if 1-xb_closed[i,0]<w:
            cost = cost+1e20*(xb_closed[i,0]-1+w)
        if 1-xb_closed[i,1]<w:
            cost = cost+1e20*(xb_closed[i,1]-1+w)

    #Distance from points in domain
    for i in range(Nd):
        v = np.zeros(Nb)
        dist0 = np.zeros(Nb)
        dmin = 1e20
        for j in range(Nb):
            #Minimum distance from each section of the road
            dist1 = dist(xb_closed[j,:], xb_closed[j+1,:],xd[i,:])
            dist2 = dist(xb_closed[j+1,:], xb_closed[j,:],xd[i,:])
            dist = max(dist1,dist2)
```

```
            dmin = min(dmin,dist)
            if dist<=w:
                v[j] = 1
                dist0[j] = dist

        #penalty if 2 or more non consecutive road segments are close to same point
        if len(np.where(np.diff(v)))>2:
            cost = cost+(w-np.sum(dist0(v==1))/np.sum(v))/w
        #penalty if no road segment is close to a point
        if dmin>w:
            cost = cost+(dmin-w)/w

    #penalty for sharp turns
    cb = 0.5*(1+cosangle(xb_closed[Nb,:],xb_closed[0,:],xb_closed[1,:]))
    if cb>0.5:
        cost = cost+5*(cb-0.5)/0.5
    for i in range(1,Nb):
        cb = 0.5*(1+cosangle(xb_closed[i-1,:],xb_closed[i,:],xb_closed[i+1,:]))
        if cb>0.5:
            cost = cost+5*(cb-0.5)/0.5
    return cost

def dist(x1,x2,x3):
    #Least distance of x3 from line segment x12
    d = 0
    x32 = x3-x2
    x12 = x1-x2
    l32 = np.linalg.norm(x32)
    l12 = np.linalg.norm(x12);
    if (l32 == 0) or (l12 == 0):
        d = 0
    else:
        cosx2 = np.dot(x32,x12)/(l32*l12)
    if cosx2<=0:
        d = l32
    else:
        d = l32*np.sqrt(1-cosx2**2)
    return d

def cosangle(x1,x2,x3):
    #angle between 2 vectors
    x12 = x1-x2
    x32 = x3-x2
    cosx2 = np.dot(x12,x32)/(np.linalg.norm(x12)*np.linalg.norm(x32))
    return cosx2

def circle_points(c,r,n):
    return [[c[0]+np.cos(2*pi/n*x)*r,c[1]+np.sin(2*pi/n*x)*r] for x in range(0,n+1)]


xd = np.random.random((100,2))
Nd = 100
w = 0.1

xcir = circle_points([0.5,0.5],0.4,20)[:20]
Nr = 20

x_sq = scipy.optimize.minimize(cost, xcir, args=(Nr,xd2,Nd,w), method='Nelder-Mead')
```

```
xf = np.reshape(x_2nd,(Nr,2))

xf1 = list(xf)
xf1.append(xf1[0])
xc1 = list(xcir)
xc1.append(xc1[0])

plt.plot(np.array(xf1)[:,0],np.array(xf1)[:,1],'-o',label='Final path')
plt.plot(np.array(xc1)[:,0],np.array(xc1)[:,1],'-o',label='Original path')
plt.plot(xd1[:,0],xd1[:,1],'.',label='Parking spots')
plt.legend(bbox_to_anchor=(0.3, 1.05))
plt.show()
```

Appendix C: Road for parking space maximization

```
xd2 = []
for i in range(3):
    xd2.append([0.5-i*0.15-0.075,1-0.075])
    xd2.append([0.5-i*0.15-0.075,0.075])
    xd2.append([0.5+i*0.15+0.075,1-0.075])
    xd2.append([0.5+i*0.15+0.075,0.075])
    xd2.append([1-0.075,0.5-i*0.15-0.075])
    xd2.append([0.075,0.5-i*0.15-0.075])
    xd2.append([1-0.075,0.5+i*0.15+0.075])
    xd2.append([0.075,0.5+i*0.15+0.075])
xd2.append([0.5-0.075,0.5-0.075])
xd2.append([0.5+0.075,0.5-0.075])
xd2.append([0.5-0.075,0.5+0.075])
xd2.append([0.5+0.075,0.5+0.075])

xd2 = np.array(xd2)
plt.plot(xd2[:,0],xd2[:,1],'.')
Nd2 = len(xd2)
w = 0.1

xcir = PointsInCircum([0.5,0.5],0.4,20)[:20]
Nr = 20

x_sq = scipy.optimize.minimize(cost, xcir, args=(Nr,xd2,Nd2,w), method='Nelder-Mead')

xf = np.reshape(x_2nd,(Nr,2))

xf1 = list(xf)
xf1.append(xf1[0])
xc1 = list(xcir)
xc1.append(xc1[0])

plt.plot(np.array(xf1)[:,0],np.array(xf1)[:,1],'-o',label='Final path')
plt.plot(np.array(xc1)[:,0],np.array(xc1)[:,1],'-o',label='Original path')
plt.plot(xd2[:,0],xd2[:,1],'.',label='Parking spots')
plt.plot([0.2,0.8,0.8,0.2,0.2],[0.2,0.2,0.8,0.8,0.2],label='Ideal path')
plt.legend(bbox_to_anchor=(0.3, 1.05))
plt.show()
```