Pricing Mechanism Design for Parking Problem

Wenfei Tang, Rohan Ghuge

December 2019

1 Introduction

In our project, we model the parking allocation problem with rational agents who tries to maximize their own utility, and we propose three different pricing mechanisms for a parking lot. We simulate these mechanisms under with varied number of agents, number of slots and probabilistic models, and decide the best pricing mechanism for the parking lot based on the result.

The report proceeds in four main sections. The first section will discuss the background information and the real-world challenge of the parking allocation problems. The second section will discuss the model we use based on the background information discussed in the first section. The third section will discuss the results of our simulation and analysis of those results. The fourth section will wrap up our report and provide some future insights on this problem. We also have an appendix of the simulation code at the very end of the report.

2 Background Information

In recent years, more and more parking lots arise with the increasing number of cities and people who own a car, leading to the growing importance of how to design the pricing mechanism for the parking lots.

The objective of a system designer is to set up a price mechanism for available parking slots. We mainly analyze from two perspectives: the profitability of the mechanism, and the welfare of agents(whether they get assigned to a parking slot or not).

We consider a set of parking lots next within a complex with office buildings and restaurants. Since the majority of our customers are white collars working in the office buildings, daily permits are sold in this parking lot. Each agent pays a price for the usage of this parking lot for the whole day.

Our agents come to the building complex for work, so they prefer to park closer to the park near their office buildings, which is their primary goal. They have their own valuation of the parking slots and will make choice based on their valuation. If none of the parking slots are available or an agent fails to find a satisfying place to park his car, he will drive to a farther but cheaper and much bigger parking lot as his default choice.

3 Model and General Structures

In this section, we provide a detailed description of our model and how we run our simulation. We assume that there are 5 office buildings(goals) G_1, G_2, G_3, G_4, G_5, and 3 parking lots S_1, S_2, S_3 within the complex, plus a default parking lot away from the complex.

1. **Basic Variables**

   (a) N: the total number of agents who visit the complex daily.

   (b) M: the total number of parking slots.

   (c) α: $\frac{N}{M}$. We use this to denote the density of each parking slot. If $N > M$, then $\alpha > 1$, otherwise $\alpha \leq 1$.

   (d) g_i: the office building where agent i works, as his goal.

   (e) $\succeq_i$: agent i has a preference for parking slots, dependent on g_i.

2. **Informational Assumptions**

   (a) *Known Goal*: g_i is known, and therefore we also know $\succeq_i$. Suppose each agent has a chip in his car to identify which building he works at.

   (b) *Known Population*: N and $\succeq_i$ are known. Suppose we have evaluated the number of agents beforehand and N is public information.

   (c) *Rational Individuals*: Suppose each agent is rational in that they make their parking choices based solely on their valuation scheme.

   (d) *Uniform Values*: For the ease of our simulation, we suppose each agent only cares about which parking lot he parks, but is indifferent about which slot he parks at within that parking lot. i.e. Each agent has the same valuation for any parking slot within the same parking lot.

3. **Valuation Schemes and Corresponding Mechanisms**
   We propose three valuation schemes as below. Each agent j w.r.t any slot s_i has a monotone decreasing function $v_j(s_i) = \phi(d(g_j, s_i))$. Intuitively, this means he always prefers a parking lot close to his office. Each agent will make his choices solely dependent on valuation, and he does not explicitly take into the payment he makes for parking at a certain slot.

   (a) *Max Distance*: In this scheme, each agent is only willing to park at the preferred parking lot withing the complex. Agent j only has one parameter m_j, representing the maximum distance agent j is willing to walk from his office building to the parking lot. Suppose agent j is allocated a slot $s_i \in S$. If $d(g_j, s_i) > m_j$, then agent j will not park in the complex's parking lots, but park at the default parking lot away from the complex.
   Agent j pays \$20 for his preferred parking lot, and pays \$5 for parking at the default parking lot.

(b) *Varied Cost*: In this scheme, each agent is willing to park at other less preferred parking lot within the complex if his preferred one is full. Agent j parks at his preferred parking slot and pays \$20; if he parks at a less preferred parking lot but still in the complex, he pays \$15. Otherwise he pays \$5 for parking at the default parking lot.

(c) *Reserved Parking with Varied Cost*: Similar to the varied cost, agent j parks at his preferred parking slot and pays \$20; if he parks at a less preferred parking lot but still in the complex, he pays \$15. Otherwise he pays \$5 for parking at the default parking lot.
The only difference is that each parking lot within the complex has a fixed ratio r of reserved parking lot specifically for agents who prefer to park here. The ratio r may vary.

4. **Measuring Metrics**
Total Profit: Each agent makes a payment t_i, and total profit $P = \sum_{i=1}^{N} t_i$. This is primary metric we use as the system designer. We would like to maximize the profit of all parking lots within the complex.

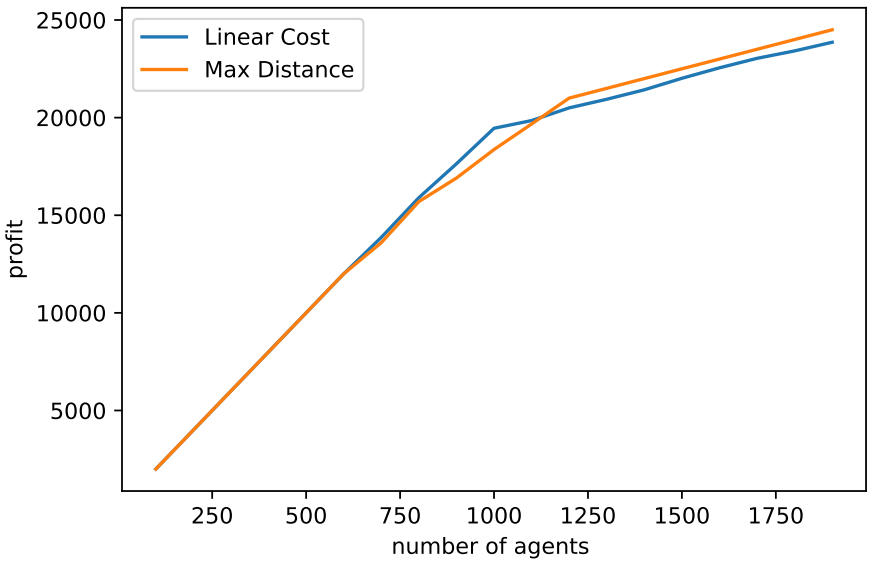
4 Simulation and Analysis

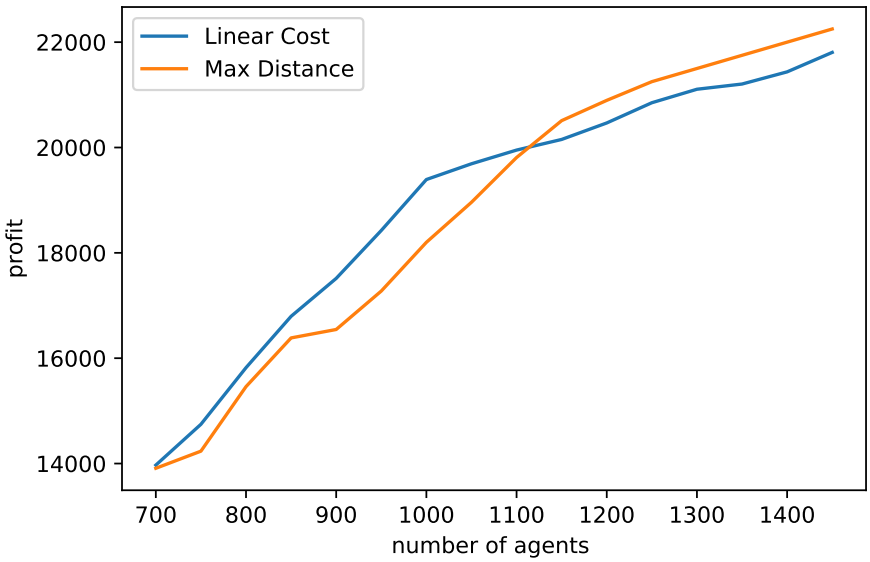
4.1 Initialization

We initialize N agents and M parking slots. Of all N agents, agent i has a random goal g_i generated uniformly from {0, 1, 2, 3, 4}, and a random max distance m_i generated uniformly from $[m_{lb}, m_{up}]$ (a pre-determined lower bound and upper bound of m_i). Each agent's distance from their goal to any parking lot is also generated uniformly from a range. Since the data is mostly generated randomly, we run 50 iterations under each setting, and then compute the average profit to filter the noise caused by random initialization.

4.2 Varied density

As we have defined in the last section, we define a new variable called *density*, denoted as $\alpha = \frac{N}{M}$. In order to explore how *density* affects the parking lot profit, we fix $M = 1000$, and vary α by varying N from 200 to 2,000. The two valuation schemes we try to compare here are *Max Distance* and *Varied Cost*. Below are the two plots from this simulation:





As we can see from the first plot, the two curves do not diverge until $N > 700$. We can conclude that the parking lot owner is indifferent about which pricing mechanism to choose when $N \leq 700$, i.e. $\alpha \leq 0.7$. Note that all the "Varied Cost" was denoted as "Linear Cost" in the plot.

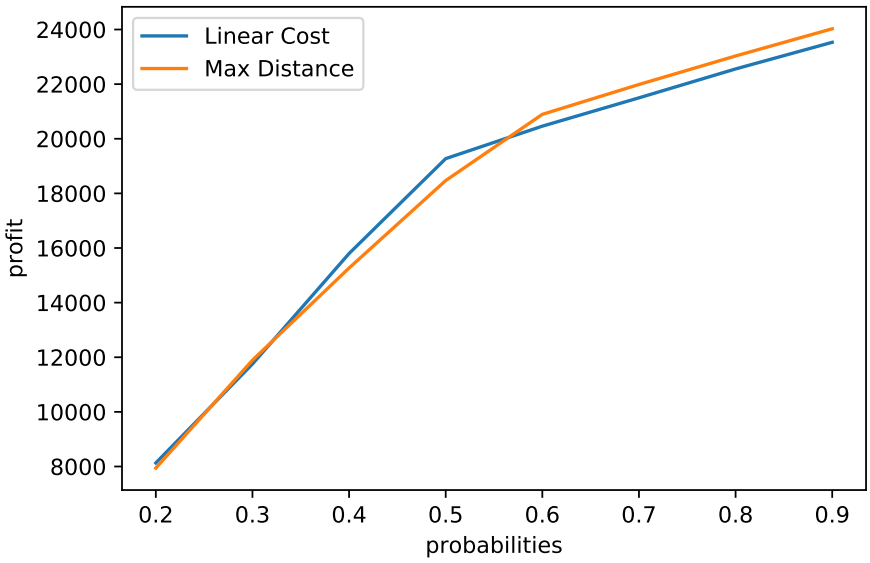
The second plot is a zoom-in of the first plot. The Varied Cost(denoted as "Linear Cost" in the plot) Mechanism outperforms Max Distance when $0.7 < \alpha < 1.1$, and Max Distance outperforms Linear Cost when $1.1 < \alpha < 2$.

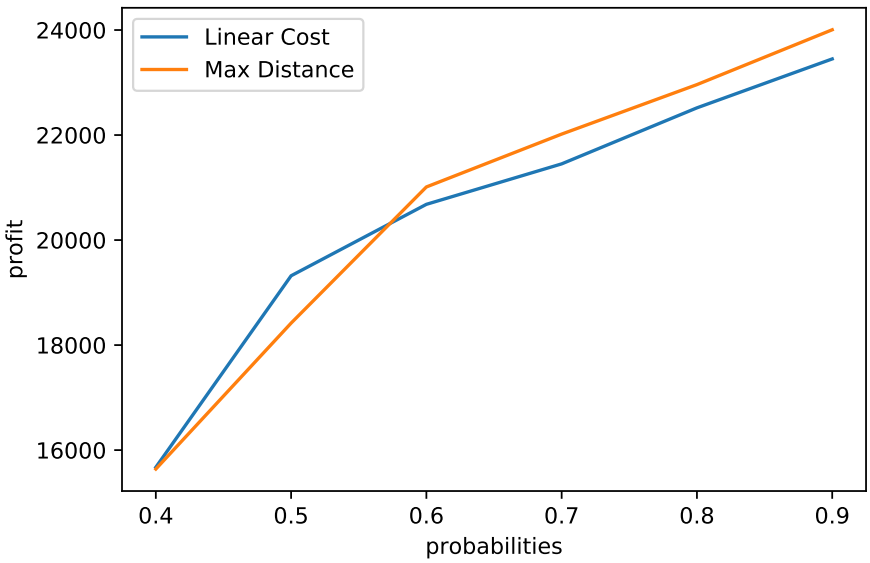
This gives a good indication to the parking lot system designer on which mechanism to choose depending on α. Given this is a complex of building with mostly office buildings, N

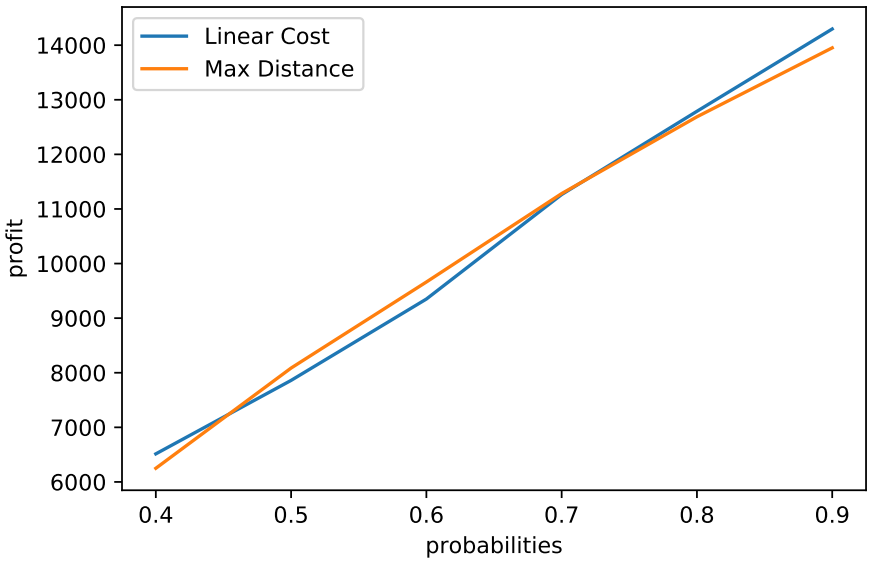
might decrease dramatically on weekends compared to weekdays. According to our simulation results, it is better to adopt Varied Cost when $\alpha < 1.1$ and adopt Max Distance when $\alpha > 1.1$. Therefore, the system designer should consider using varied Cost on weekends and max distance on weekdays, given there more people who need a parking slot on weekdays.

4.3 Probabilistic model

Based on our analysis in section 4.2, we notice that whether an agent will come to the complex is a deterministic factor of which mechanism to use. Therefore, we introduce a probability function p_i for each agent i to model their possibility of coming to the complex buildings and needing to park their cars. Here, we fix the $M = 1000$ parking slots and $N = 800$ and $N = 2000$ relatively, with varying probability function p_i randomly drawn from 0.2 to 0.9. Below are the three plots we obtain from this simulation:







The first two plots are generated given that $M = 1000$, $N = 2000$. And the second plot is a zoom-in of the first plot.

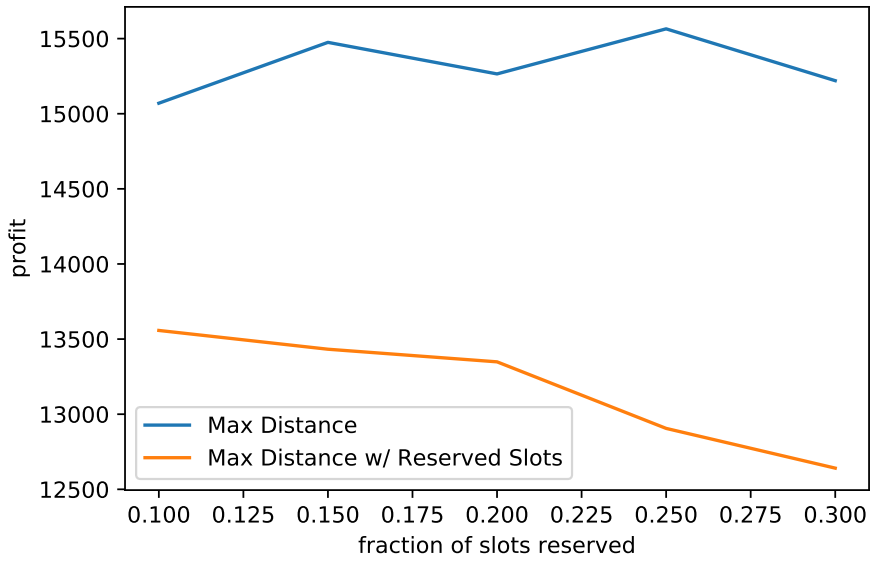
In the first simulation, we notice that when N ¡ 700, the profit of two mechanisms are almost identical. Similar to the results from the first simulation, we can see that the system designer could be indifferent about which mechanism to use given a low probability that the agents come to the complex is below 0.4. The third plot, which is generated from $M = 1000$, $N = 800$, further supports this claim, as the two curves in this plot almost overlap.

4.4 Reserved Parking

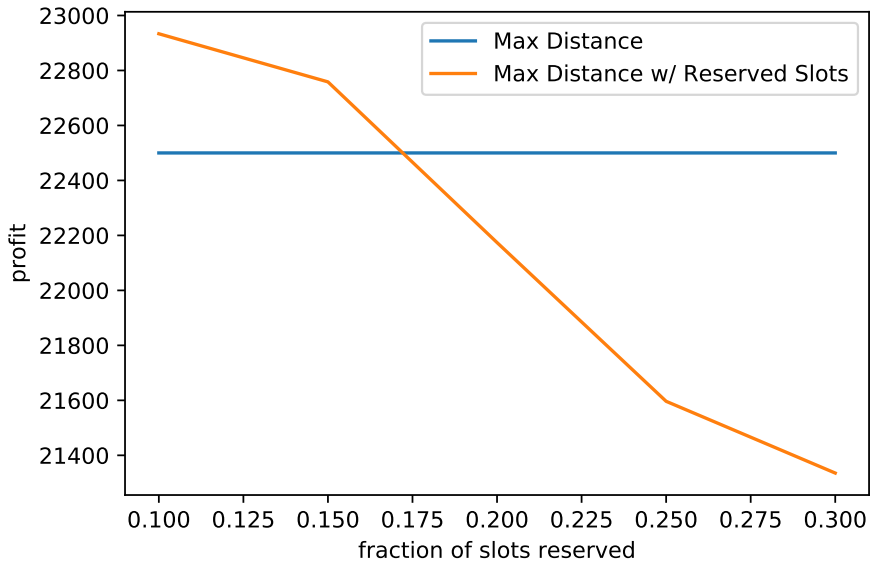
In this simulation, we would like to focus on a new mechanism–reserved parking. The basic idea is that there are a certain amount of reserved parking slots for agents who prefer to park here(and also willing to pay more as they pay \$20 rather than \$15 or \$5). Many parking lots adopt this mechanism. For example, University of Michigan issues blue parking permits and yellow parking permits for reserved and unreserved parking slots. We will look at Max Distance and Varied Cost separately in the below simulations. Here we fix $M = 1000$, $N = 800$ and $N = 800$ relatively, with varied fraction of slots reserved.

4.4.1 Max Distance

1. $\alpha < 1$, $N = 800 < M$.



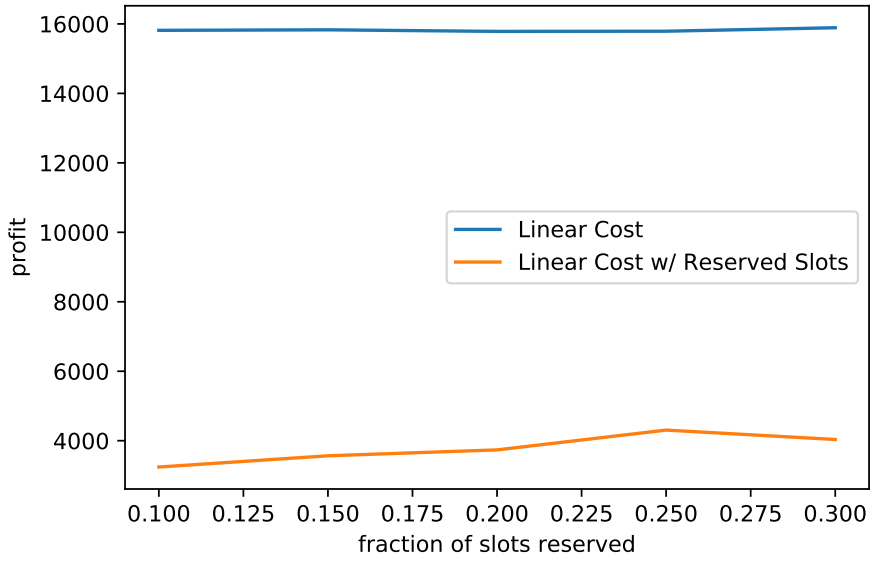
2. $\alpha > 1$, $N = 1500 > M$.



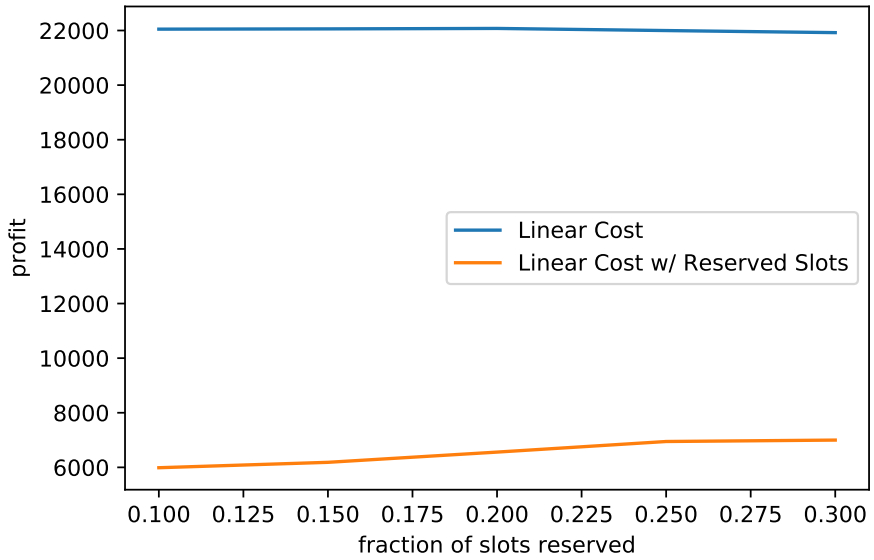
When simulating with Max Distance Mechanism, Max Distance without reservation outperforms Max Distance with Reserved Slots in most cases. There is only an exception when $N = 1500 > M$ and fraction $r < 0.175$. When there are more agents than parking slots, the system designer should consider reserve a small fraction, say $r = 0.15$ of parking slots as reserved parking, leading to a boost in profit.

4.4.2 Varied Cost

1. $\alpha < 1$, $N = 800 < M$.



2. $\alpha > 1$, $N = 1500 > M$.



When simulating with Varied Cost Mechanism, Varied Cost without reservation always outperforms Varied Cost with reservation. So there is no point in adopting reservation if we use Varied Cost as the pricing mechanism.

5 Conclusion and Future Work

In our project, we design a simplified model With the goal of maximizing the profit of the parking structure in the complex and at the same time trying to satisfy each agent's preference as much as possible. This model allows us to run relevant simulations and contributes to our understanding of how different valuation schemes and pricing mechanisms would affect agents' behavior and the total profit. Furthermore, the results of these simulations provide useful insights for the system designers who want to find a reasonable and practical pricing mechanisms for the parking lot system.

While our model and simulations provide some suggestive insights for system designers, it is undeniable that there are several limitations of our work as addressed below.

1. The valuation scheme and pricing systems we discuss here are very straight forward. Even though we do not want a very complex pricing system for parking lots in real life, there are still room to improve the current scheme. For example, the varied cost scheme is in fact a pseudo linear cost scheme–the agent walks further and pays less. Due to the complexity of the linear cost scheme and the lack of data set for us to test, we adopt a relatively simple varied cost scheme. However, we do think the linear cost is a scheme worth discussing.

2. In this simplified model, we assume that the agents as white collars who work at the office buildings will only purchase daily permits in these parking lots and will occupy

the parking slot for the whole day. However, the more common case is that there might also be a decent amount of agents who do not want to park their car in the slot for whole day. In addition to daily permits, a parking lot is also very likely to offer a certain fraction of flexible parking slots for agents who would come and go.

6 Appendix: Simulation Code

```
    #!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  9 23:00:35 2019

@author: rohanghuge, twenfei
"""

import numpy as np
import time
import matplotlib.pyplot as plt


# default option cost
phi = 20

# setting the structure: S, G, d
m = 1000
k = 5

# set number of agents to 3000
N = 800

# setting a permutation over the agents
pi = np.random.permutation(N)

# upper and lower bounds for max distance (can vary these)
m_lb = 200
m_ub = 500

# sample m_j values for each agent uniformly at random in range (200, 500, 50)
max_distances = np.round(np.random.uniform(200, 500, N))

# set goals for each agent at random
elements = [0, 1, 2, 3, 4]
probabilities = [0.3, 0.2, 0.15, 0.15, 0.2]
#probabilities = [0.2, 0.2, 0.2, 0.2, 0.2]
```

```
goals = np.random.choice(elements, N, p=probabilities)


# prefernce of an agent based on max-distance
def pref(j, s):
    if 0 <= s < 300:
        if goals(j) == 1 or goals(j) == 2:
            return phi+5
        else: return 0

    if 300 <= s < 400:
        if goals(j) == 3 or goals(j) == 4:
            return phi+5
        else: return 0

    if 400 <= s < 500:
        if goals(j) == 5:
            return phi+5
        else: return 0
    return 0


# begin simulation
maxdist_profit = []
lin_profit = []

maxdist_profit_res = []
lin_profit_res = []

N = 1500
prob=1
#for N in range(100, 2000, 100):
#for prob in [0.4, 0.5, 0.6, 0.7, 0.8, 0.9]:
for res_prob in [0.1, 0.15, 0.2, 0.25, 0.3]:
    print(res_prob)

    # set goals for each agent at random
    elements = [0, 1, 2, 3, 4]
    probabilities = [0.3, 0.2, 0.15, 0.15, 0.2]
    #probabilities = [0.2, 0.2, 0.2, 0.2, 0.2]
    goals = np.random.choice(elements, N, p=probabilities)

    days = 10
    price = 20
    soc_welfare = np.zeros(days)
```

```
profit = np.zeros(days)
slots_filled = np.zeros(days)

for i in range(days):

    matchings = np.zeros(N)
    pi = np.random.permutation(N)
    slots = np.zeros(m)

    for agent in pi:

        shows_up = np.random.binomial(1, prob)
        if shows_up == 0:
            continue

        #print('Agent ', agent, ' arrived')
        goal = goals[agent]
        matched = 0
        s = 'Agent ' + str(agent) + ' arrived.'
        #print(s)

        if goal == 0 or goal == 1:
            for k in range(int(0.6*m)):
                if slots[k] == 0 and matched == 0:
                    matchings[agent] = k
                    slots[k] = 1
                    profit[i] = profit[i] + price
                    matched = 1
                    s = 'Agent ' + str(agent) + ' found a slot.'
                    #print(s)
            if matched == 0:
                s = 'Agent ' + str(agent) + ' did not find a slot.'
                #print(s)
                profit[i] = profit[i] + 5

        if goal == 2 or goal == 3:
            for k in range(int(0.6*m), int(0.8*m)):
                if slots[k] == 0 and matched == 0:
                    matchings[agent] = k
                    slots[k] = 1
                    profit[i] = profit[i] + price
                    matched = 1
                    s = 'Agent ' + str(agent) + ' found a slot.'
                    #print(s)
            if matched == 0:
```

```
                s = 'Agent ' + str(agent) + ' did not find a slot.'
                #print(s)
                profit[i] = profit[i] + 5

        if goal == 4:
            for k in range(int(0.8*m), m):
                if slots[k] == 0 and matched == 0:
                    matchings[agent] = k
                    slots[k] = 1
                    profit[i] = profit[i] + price
                    matched = 1
                    s = 'Agent ' + str(agent) + ' found a slot.'
                    #print(s)
            if matched == 0:
                s = 'Agent ' + str(agent) + ' did not find a slot.'
                #print(s)
                profit[i] = profit[i] + 5

    slots_filled[i] = sum(slots)

maxdist_profit = maxdist_profit + [np.mean(profit)]


# begin simulation 2 - variable prices
days = 10
price = 20
soc_welfare2 = np.zeros(days)
profit2 = np.zeros(days)
slots_filled2 = np.zeros(days)


for i in range(days):

    matchings2 = np.zeros(N)
    pi = np.random.permutation(N)
    slots = np.zeros(m)

    for agent in pi:

        shows_up = np.random.binomial(1, prob)
        if shows_up == 0:
            continue

        #print('Agent ', agent, ' arrived')
        goal = goals[agent]
```

```
matched = 0
s = 'Agent ' + str(agent) + ' arrived.'
#print(s)

if goal == 0 or goal == 1:
    for k in range(int(0.6*m)):
        if slots[k] == 0 and matched == 0:
            matchings[agent] = k
            slots[k] = 1
            profit2[i] = profit2[i] + price
            matched = 1
            s = 'Agent ' + str(agent) + ' found a slot.'
            #print(s)

    for k in range(int(0.6*m), m):
        if slots[k] == 0 and matched == 0:
            matchings[agent] = k
            slots[k] = 1
            profit2[i] = profit2[i] + price - 5
            # offer cheaper price to agent for other parking
            matched = 1
            s = 'Agent ' + str(agent) + ' found a slot.'
            #print(s)

    if matched == 0:
        s = 'Agent ' + str(agent) + ' did not find a slot.'
        #print(s)
        profit2[i] = profit2[i] + 5

if goal == 2 or goal == 3:
    for k in range(int(0.6*m), int(0.8*m)):
        if slots[k] == 0 and matched == 0:
            matchings[agent] = k
            slots[k] = 1
            profit2[i] = profit2[i] + price
            matched = 1
            s = 'Agent ' + str(agent) + ' found a slot.'
            #print(s)

    for k in range(int(0.6*m)):
        if slots[k] == 0 and matched == 0:
            matchings[agent] = k
            slots[k] = 1
            profit2[i] = profit2[i] + price - 5
            # offer cheaper price to agent for other parking
```

```
                matched = 1
                s = 'Agent ' + str(agent) + ' found a slot.'
                #print(s)

        for k in range(int(0.8*m), m):
            if slots[k] == 0 and matched == 0:
                matchings[agent] = k
                slots[k] = 1
                profit2[i] = profit2[i] + price - 5
                # offer cheaper price to agent for other parking
                matched = 1
                s = 'Agent ' + str(agent) + ' found a slot.'
                #print(s)



        if matched == 0:
            s = 'Agent ' + str(agent) + ' did not find a slot.'
            #print(s)
            profit2[i] = profit2[i] + 5

    if goal == 4:
        for k in range(int(0.8*m), m):
            if slots[k] == 0 and matched == 0:
                matchings[agent] = k
                slots[k] = 1
                profit2[i] = profit2[i] + price
                matched = 1
                s = 'Agent ' + str(agent) + ' found a slot.'
                #print(s)

        for k in range(int(0.8*m)):
            if slots[k] == 0 and matched == 0:
                matchings[agent] = k
                slots[k] = 1
                profit2[i] = profit2[i] + price - 5
                # offer cheaper price to agent for other parking
                matched = 1
                s = 'Agent ' + str(agent) + ' found a slot.'
                #print(s)

        if matched == 0:
            s = 'Agent ' + str(agent) + ' did not find a slot.'
            #print(s)
            profit2[i] = profit2[i] + 5
```

```
    slots_filled2[i] = sum(slots)

lin_profit = lin_profit + [np.mean(profit2)]

days = 10
price = 20
exp_price = 40
wtp = 0.2 # prob that an agent will purchase reserved parking

profit = np.zeros(days)
slots_filled = np.zeros(days)

for i in range(days):

    matchings = np.zeros(N)
    pi = np.random.permutation(N)
    slots = np.zeros(m)

    for agent in pi:

        shows_up = np.random.binomial(1, prob)
        if shows_up == 0:
            continue

        #print('Agent ', agent, ' arrived')
        goal = goals[agent]
        matched = 0
        s = 'Agent ' + str(agent) + ' arrived.'
        #print(s)

        if goal == 0 or goal == 1:
            for k in range(int(0.6*m-res_prob*0.6*m)):
                if slots[k] == 0 and matched == 0:
                    matchings[agent] = k
                    slots[k] = 1
                    profit[i] = profit[i] + price
                    matched = 1
                    s = 'Agent ' + str(agent) + ' found a slot.'
                    #print(s)

            willing_to_pay = np.random.binomial(1, wtp)
            if matched == 0 and willing_to_pay == 1:
                for k in range(int(0.6*m-res_prob*0.6*m)+1, int(0.6*m)):
                    if slots[k] == 0 and matched == 0:
```

```
                matchings[agent] = k
                slots[k] = 1
                profit[i] = profit[i] + exp_price
                matched = 1
                s = 'Agent ' + str(agent) + ' found a slot.'
                #print(s)


    if matched == 0:
        s = 'Agent ' + str(agent) + ' did not find a slot.'
        #print(s)
        profit[i] = profit[i] + 5


if goal == 2 or goal == 3:
    for k in range(int(0.6*m), int(0.8*m)-int(0.8*m-res_prob*0.8*m)):
        if slots[k] == 0 and matched == 0:
            matchings[agent] = k
            slots[k] = 1
            profit[i] = profit[i] + price
            matched = 1
            s = 'Agent ' + str(agent) + ' found a slot.'
            #print(s)

    willing_to_pay = np.random.binomial(1, wtp)
    if matched == 0 and willing_to_pay == 1:
        for k in range(int(0.8*m-res_prob*0.8*m)+1, int(0.8*m)):
            if slots[k] == 0 and matched == 0:
                matchings[agent] = k
                slots[k] = 1
                profit[i] = profit[i] + exp_price
                matched = 1
                s = 'Agent ' + str(agent) + ' found a slot.'
                #print(s)


    if matched == 0:
        s = 'Agent ' + str(agent) + ' did not find a slot.'
        #print(s)
        profit[i] = profit[i] + 5

if goal == 4:
    for k in range(int(0.8*m), m - int(res_prob*0.8*m)):
        if slots[k] == 0 and matched == 0:
            matchings[agent] = k
```

```
                    slots[k] = 1
                    profit[i] = profit[i] + price
                    matched = 1
                    s = 'Agent ' + str(agent) + ' found a slot.'
                    #print(s)
            willing_to_pay = np.random.binomial(1, wtp)
            if matched == 0 and willing_to_pay == 1:
                for k in range(m - int(res_prob*0.8*m)+1, m):
                    if slots[k] == 0 and matched == 0:
                        matchings[agent] = k
                        slots[k] = 1
                        profit[i] = profit[i] + exp_price
                        matched = 1
                        s = 'Agent ' + str(agent) + ' found a slot.'
                        #print(s)



            if matched == 0:
                s = 'Agent ' + str(agent) + ' did not find a slot.'
                #print(s)
                profit[i] = profit[i] + 5

    slots_filled[i] = sum(slots)

maxdist_profit_res = maxdist_profit_res + [np.mean(profit)]


# begin simulation 2 - variable prices
days = 10
price = 20
soc_welfare2 = np.zeros(days)
profit2 = np.zeros(days)
slots_filled2 = np.zeros(days)

# the agent is less likely to pay for
# reserved slot since she can try to park in another parking lot
wtp2 = wtp


for i in range(days):

    matchings2 = np.zeros(N)
    pi = np.random.permutation(N)
    slots = np.zeros(m)
```

```
for agent in pi:

    shows_up = np.random.binomial(1, prob)
    if shows_up == 0:
        continue

    #print('Agent ', agent, ' arrived')
    goal = goals[agent]
    matched = 0
    s = 'Agent ' + str(agent) + ' arrived.'
    #print(s)

    if goal == 0 or goal == 1:
        for k in range(int(0.6*m-res_prob*0.6*m)):
            if slots[k] == 0 and matched == 0:
                matchings[agent] = k
                slots[k] = 1
                profit[i] = profit[i] + price
                matched = 1
                s = 'Agent ' + str(agent) + ' found a slot.'
                #print(s)

        willing_to_pay = np.random.binomial(1, wtp2)
        if matched == 0 and willing_to_pay == 1:
            for k in range(int(0.6*m-res_prob*0.6*m)+1, int(0.6*m)):
                if slots[k] == 0 and matched == 0:
                    matchings[agent] = k
                    slots[k] = 1
                    profit[i] = profit[i] + exp_price
                    matched = 1
                    s = 'Agent ' + str(agent) + ' found a slot.'
                    #print(s)


        for k in range(int(0.6*m), int(0.8*m)-int(0.8*m-res_prob*0.8*m)):
            if slots[k] == 0 and matched == 0:
                matchings[agent] = k
                slots[k] = 1
                profit2[i] = profit2[i] + price - 5
                # offer cheaper price to agent for other parking
                matched = 1
                s = 'Agent ' + str(agent) + ' found a slot.'
                #print(s)
```

```
        for k in range(int(0.8*m), m - int(res_prob*0.8*m)):
            if slots[k] == 0 and matched == 0:
                matchings[agent] = k
                slots[k] = 1
                profit2[i] = profit2[i] + price - 5
                # offer cheaper price to agent for other parking
                matched = 1
                s = 'Agent ' + str(agent) + ' found a slot.'
                #print(s)


        if matched == 0:
            s = 'Agent ' + str(agent) + ' did not find a slot.'
            #print(s)
            profit2[i] = profit2[i] + 5

    if goal == 2 or goal == 3:
        for k in range(int(0.6*m), int(0.8*m)-int(0.8*m-res_prob*0.8*m)):
            if slots[k] == 0 and matched == 0:
                matchings[agent] = k
                slots[k] = 1
                profit[i] = profit[i] + price
                matched = 1
                s = 'Agent ' + str(agent) + ' found a slot.'
                #print(s)

        willing_to_pay = np.random.binomial(1, wtp)
        if matched == 0 and willing_to_pay == 1:
            for k in range(int(0.8*m-res_prob*0.8*m)+1, int(0.8*m)):
                if slots[k] == 0 and matched == 0:
                    matchings[agent] = k
                    slots[k] = 1
                    profit[i] = profit[i] + exp_price
                    matched = 1
                    s = 'Agent ' + str(agent) + ' found a slot.'
                    #print(s)

        for k in range(int(0.6*m-res_prob*0.6*m)):
            if slots[k] == 0 and matched == 0:
                matchings[agent] = k
                slots[k] = 1
                profit2[i] = profit2[i] + price - 5
                # offer cheaper price to agent for other parking
                matched = 1
                s = 'Agent ' + str(agent) + ' found a slot.'
```

```
                #print(s)

        for k in range(int(0.8*m), m - int(res_prob*0.8*m)):
            if slots[k] == 0 and matched == 0:
                matchings[agent] = k
                slots[k] = 1
                profit2[i] = profit2[i] + price - 5
                # offer cheaper price to agent for other parking
                matched = 1
                s = 'Agent ' + str(agent) + ' found a slot.'
                #print(s)



        if matched == 0:
            s = 'Agent ' + str(agent) + ' did not find a slot.'
            #print(s)
            profit2[i] = profit2[i] + 5

    if goal == 4:
        for k in range(int(0.8*m), m - int(res_prob*0.8*m)):
            if slots[k] == 0 and matched == 0:
                matchings[agent] = k
                slots[k] = 1
                profit[i] = profit[i] + price
                matched = 1
                s = 'Agent ' + str(agent) + ' found a slot.'
                #print(s)
        willing_to_pay = np.random.binomial(1, wtp)
        if matched == 0 and willing_to_pay == 1:
            for k in range(m - int(res_prob*0.8*m)+1, m):
                if slots[k] == 0 and matched == 0:
                    matchings[agent] = k
                    slots[k] = 1
                    profit[i] = profit[i] + exp_price
                    matched = 1
                    s = 'Agent ' + str(agent) + ' found a slot.'
                    #print(s)

        for k in range(int(0.6*m-res_prob*0.6*m)):
            if slots[k] == 0 and matched == 0:
                matchings[agent] = k
                slots[k] = 1
                profit2[i] = profit2[i] + price - 5
                # offer cheaper price to agent for other parking
```

```
                        matched = 1
                        s = 'Agent ' + str(agent) + ' found a slot.'
                        #print(s)

                for k in range(int(0.6*m), int(0.8*m)-int(0.8*m-res_prob*0.8*m)):
                    if slots[k] == 0 and matched == 0:
                        matchings[agent] = k
                        slots[k] = 1
                        profit2[i] = profit2[i] + price - 5
                        # offer cheaper price to agent for other parking
                        matched = 1
                        s = 'Agent ' + str(agent) + ' found a slot.'
                        #print(s)


                if matched == 0:
                    s = 'Agent ' + str(agent) + ' did not find a slot.'
                    #print(s)
                    profit2[i] = profit2[i] + 5

        slots_filled2[i] = sum(slots)

    lin_profit_res = lin_profit_res + [np.mean(profit2)]



x = [0.1, 0.15, 0.2, 0.25, 0.3]


# plotting the points
plt.plot(x, lin_profit, label='Linear Cost')
#plt.plot(x, maxdist_profit, label='Max Distance')
plt.plot(x, lin_profit_res, label='Linear Cost w/ Reserved Slots')
#plt.plot(x, maxdist_profit_res, label='Max Distance w/ Reserved Slots')

plt.xlabel('fraction of slots reserved')
plt.ylabel('profit')
plt.legend()
plt.savefig('plt8.pdf')
plt.show()
```