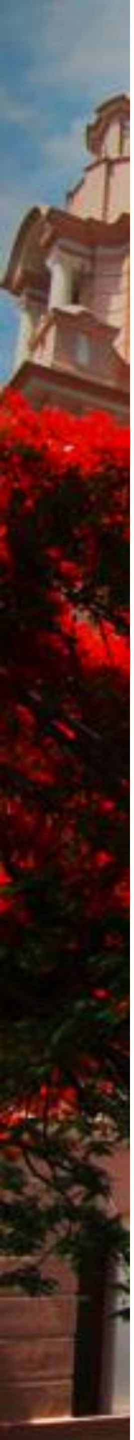
MSBA7003 Quantitative Analysis Methods

03 Monte Carlo Tree Search

Wei Zhang

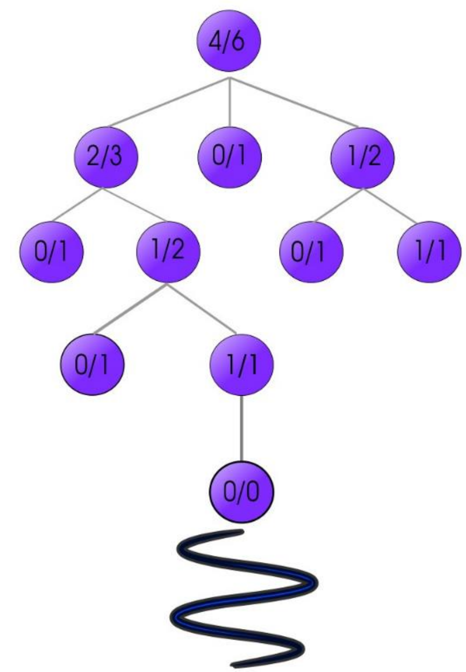
2019 - 2020

香 港 大 學
THE UNIVERSITY OF HONG KONG



Agenda

- AlphaGo and Game AI
- Monte Carlo Tree Search
  - Concepts
  - Search Strategy
  - Simulation Strategy
  - Implementation
- Carpark Revenue Management

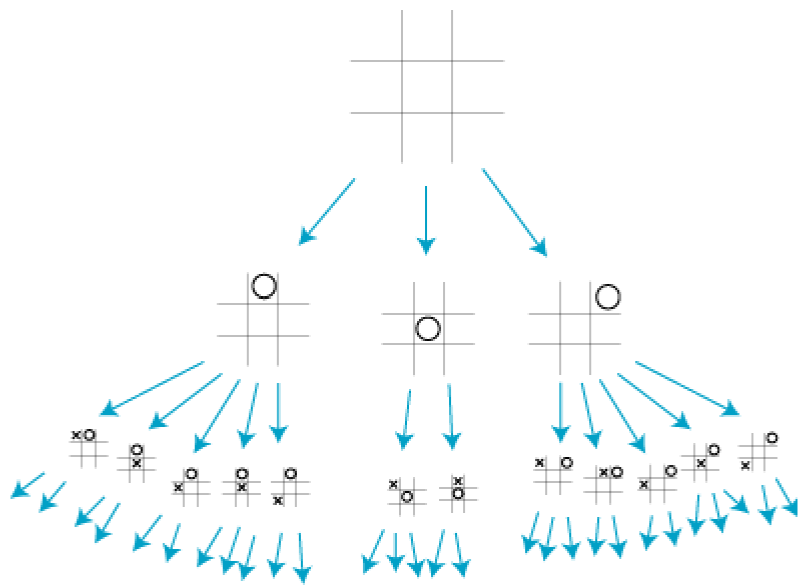


AlphaGo and Game AI

- AlphaGo won Lee Sedol and Ke Jie, the top two go players.
- Monte Carlo Tree Search (MCTS) is at the heart of AlphaGo's algorithm.

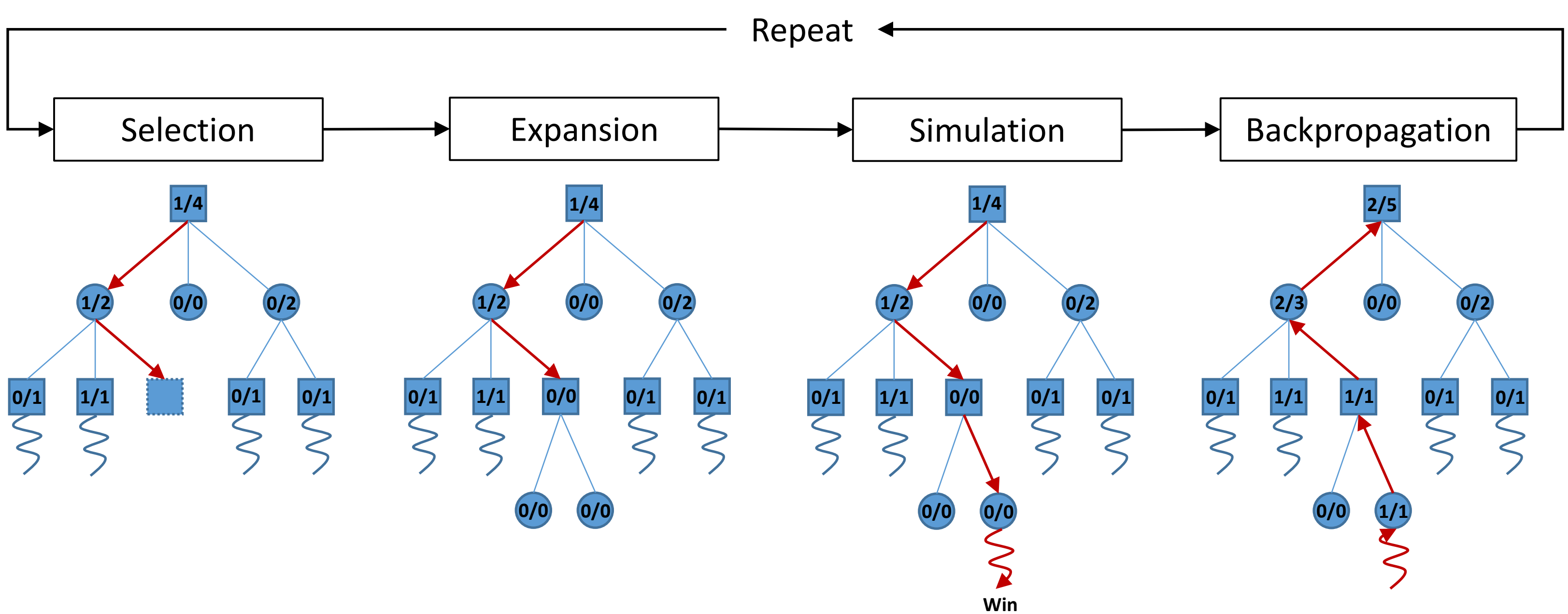
AlphaGo and Game AI

- Normally, a game can be solved by DP.
- DP fails if the game tree is too large or the random events follow complicated or unknown probability distributions.



Monte Carlo Tree Search

- The basic MCTS algorithm is to build a search tree, node by node, according to the outcomes of simulated playouts.
- The process can be broken down into four steps.



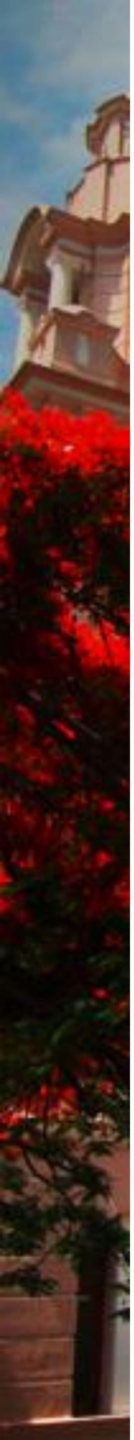
Monte Carlo Tree Search

- Initialization: Create the root node (i.e., the level 1 decision node) and the child nodes (i.e., the state-of-nature nodes) following each option.
- Selection: Search forward by selecting a state-of-nature node (or an option) according to a given strategy. After that, select or create a child node (or a state of nature) through simulation. Repeat until no more child nodes are available.
- Expansion: If the last node is not a terminal node, create all the state-of-nature nodes (i.e., all the options).
- Simulation: Repeat the previous two steps (selection & expansion) until a terminal node is reached.
- Backpropagation: Update the summary statistics of each node along the path according to the result.

Selection Strategy

- Random Search
  - Focuses on exploration (i.e., look in areas that have not been well sampled yet) but ignores exploitation (i.e., look in areas which appear to be promising)
  - The “optimal” option given by the tree in the end may not be truly optimal

- Upper Confidence Bound (UCB1) Strategy
  - Balances exploitation of the currently most promising option with exploration of alternatives which may later turn out to be better
  - Converges to optimal decisions given sufficient time



UCB1 Strategy

- The upper confidence bound for node i (an option) is given by

$$B_i = V_i + 2\sqrt{\frac{\ln N_i}{n_i}}$$

- V_i is the average total reward/value achieved by paths after node i (going forward)
- N_i is the number of times the parent node of i has been visited
- n_i is the number of times node i has been visited

- At a decision node, select the child node that maximizes the UCB.

UCB1 Strategy

- If more than one child node has the same maximum UCB value, the tie is usually broken randomly.
- The first term drives exploitation and the second term drives exploration. If a child node is selected, the value of its second term will decrease but the value will increase for other child nodes.
- $n_i = 0$ yields a UCB value of infinity, so previously unvisited child nodes are assigned the largest value in order to ensure that all child nodes are considered at least once before any child node is further expanded.
- The formula can be modified to $B_i = V_i + 2C\sqrt{\frac{\ln N_i}{n_i}}$, where C can be adjusted to lower or increase the amount of exploration performed. If the reward is beyond the range of [0,1], the value of C can be carefully chosen to achieve the balance.

Simulation Strategy

- If the state of nature is independent of previous decisions, the realization of the state can be simulated by historical data or according to certain distributions.

- If the state of nature depends on previous decisions, a model can be built to capture the dependence of the state on previous decisions.

- For games, the simulation of the opponent's move can be done by other algorithms.

Implementation: Building Search Tree

- The MCTS algorithm with UCB1 selection strategy can be implemented with a table.

Node Index	Parent Index	Child Index	Node Type	Meaning	n	V	UCB
1	0	{2,3,4}	Decision	The root node	4	1/4	-
2	1	{…}	State	Option 1-1	2	1/2	1.5973
3	1	{}	State	Option 1-2	0	0	∞
4	1	{…}	State	Option 1-3	2	0	1.0973
…							

Implementation: Building Search Tree

Node Index	Parent Index	Child Index	Node Type	Meaning	n	V	UCB
1	0	{2,3,4}	Decision	The root node	4	1/4	-
2	1	{…}	State	Option 1(a)	2	1/2	1.5973
3	1	{K+1}	State	Option 1(b)	0	0	∞
4	1	{…}	State	Option 1(c)	2	0	1.0973
…							
K + 1	3	{}	Decision	State 3(a)	0	0	∞

Implementation: Building Search Tree

Node Index	Parent Index	Child Index	Node Type	Meaning	n	V	UCB
1	0	{2,3,4}	Decision	The root node	4	1/4	-
2	1	{…}	State	Option 1(a)	2	1/2	1.5973
3	1	{K+1}	State	Option 1(b)	0	0	∞
4	1	{…}	State	Option 1(c)	2	0	1.0973
…							
K + 1	3	{K+2,K+3}	Decision	State 3(a)	0	0	∞
K + 2	K + 1	{}	State	Option (K+1)(a)	0	0	∞
K + 3	K + 1	{}	State	Option (K+1)(b)	0	0	∞

香 港 大 學
THE UNIVERSITY OF HONG KONG

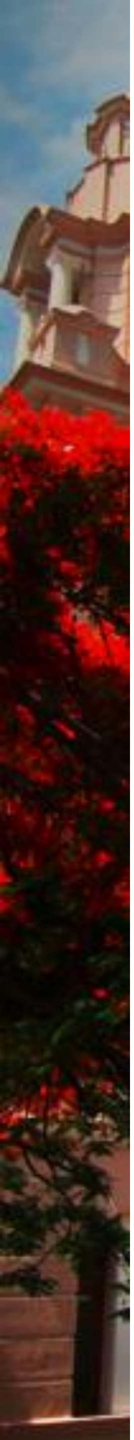
Implementation: Building Search Tree

Node Index	Parent Index	Child Index	Node Type	Meaning	n	V	UCB
1	0	{2,3,4}	Decision	The root node	5	2/5	-
2	1	{…}	State	Option 1(a)	2	1/2	1.5973
3	1	{K+1}	State	Option 1(b)	1	1	1
4	1	{…}	State	Option 1(c)	2	0	1.0973
…							
K + 1	3	{K+2,K+3}	Decision	State 3(a)	1	1	1
K + 2	K + 1	{…}	State	Option (K+1)(a)	1	1	1
K + 3	K + 1	{}	State	Option (K+1)(b)	0	0	∞
…							
K’	…	{}	Terminal	Outcome	1	1	1

Implementation: Optimization

- When doing optimization:
  - Select the option that maximizes the total reward/value V
  - The state of nature realizes according to the real situation
  - If the state does not exist in the current tree, perform the MCTS algorithm in real time and then do the optimization

- Notes:
  - The root node can be a state-of-nature node if the first decision depends on the initial state



Carpark Revenue Management

- Company Background
  - SPS is one of the largest carpark operators in Hong Kong.
  - Three types of customers
    - Hourly parking
    - Floating monthly parking
    - Reserved monthly parking
- Problem to Solve
  - 50 spaces, 20 floating monthly users, 10 reserved monthly users
  - Maximize revenue from hourly parking without affecting monthly customers
  - When to show the “FULL” sign?

Carpark Revenue Management

- Challenges:
  - Arrivals and lengths of stay for hourly customers were not recorded when the “FULL” sign is on
  - Demand can be different for different days and different hours of a day
  - Decisions made across a day are not independent. Because the length of stay is random, accepting too many hourly users may affect monthly users several hours later
- Solution:
  - Re-generate the hourly parking arrival time intervals and the lengths of stay with the uncensored data
  - Use Monte Carlo Tree Search to decide whether to show the “FULL” sign for every 15-min time interval

Carpark Revenue Management

Hourly_ReGen.csv

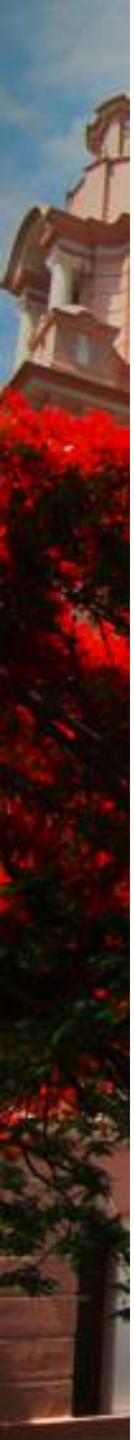
	A	B	C	D	E
1	Day	Weekday	ArrivalTime	ExitTime	LengthOfStay
2	1	1	3:14:39 AM	1:32:17 PM	0.428914812
3	1	1	7:33:36 AM	8:43:28 AM	0.048511023
4	1	1	7:53:22 AM	8:27:01 AM	0.023367123
5	1	1	8:07:14 AM	8:15:54 AM	0.006011036
6	1	1	8:09:54 AM	8:15:46 AM	0.004064706
7	1	1	8:19:32 AM	10:35:00 AM	0.094076865
8	1	1	8:22:31 AM	11:24:48 AM	0.126584474
9	1	1	8:30:30 AM	11:53:05 AM	0.140678843
10	1	1	8:31:19 AM	10:20:21 AM	0.075724936
11	1	1	8:37:13 AM	10:44:10 AM	0.088166476
12	1	1	8:38:18 AM	8:40:19 AM	0.001390789
13	1	1	8:46:40 AM	10:56:28 AM	0.090138633
14	1	1	8:50:12 AM	12:14:39 PM	0.141980597
15	1	1	8:58:50 AM	12:29:12 PM	0.146079134
16	1	1	9:08:04 AM	12:21:01 PM	0.133991657
17	1	1	9:24:31 AM	4:18:51 PM	0.287733225
18	1	1	9:31:54 AM	10:03:37 AM	0.022021927
19	1	1	10:44:19 AM	6:00:38 PM	0.302997493
20	1	1	11:19:33 AM	3:15:14 PM	0.163670044
21	1	1	11:23:31 AM	12:52:50 PM	0.062023683
22	1	1	12:12:19 PM	3:08:08 PM	0.122092646
23	1	1	12:31:53 PM	7:40:28 PM	0.297633705
24	1	1	2:11:55 PM	2:42:22 PM	0.021145938
25	1	1	2:29:16 PM	9:11:44 PM	0.279491553
26	1	1	3:36:49 PM	5:20:50 PM	0.072236857
27	1	1	3:48:28 PM	8:35:52 PM	0.199582797
28	1	1	5:10:26 PM	7:40:34 PM	0.104255116
29	1	1	6:53:01 PM	12:00:59 AM	0.213867708
30	1	1	7:42:15 PM	10:19:42 PM	0.109334532
31	1	1	8:55:43 PM	4:41:51 AM	0.323695337
32	1	1	10:08:13 PM	2:39:32 AM	0.188412102
33	1	1	10:44:15 PM	1:03:22 AM	0.096607471
34	1	1	11:40:20 PM	6:39:54 AM	0.291355921
35	2	2	12:20:12 AM	12:44:18 AM	0.016740428

MonthlyFloating_ReGen.csv

	A	B	C	D	E	F
1	Day	Weekday	CarNo	ExitTime	ReturnTime	Duration
2	1	1	5	6:02:40 AM	11:38:56 AM	0.2335212
3	1	1	1	6:05:07 AM	10:14:05 PM	0.6729025
4	1	1	16	6:08:12 AM	5:50:27 PM	0.4876705
5	1	1	14	6:27:41 AM	9:18:47 PM	0.6188215
6	1	1	13	6:35:46 AM	5:40:20 PM	0.4615014
7	1	1	6	7:01:45 AM	3:13:47 PM	0.3416949
8	1	1	7	7:10:47 AM	10:19:35 AM	0.1311133
9	1	1	12	7:11:10 AM	2:13:40 PM	0.2934063
10	1	1	18	7:25:09 AM	10:38:45 AM	0.1344443
11	1	1	3	7:26:39 AM	8:32:11 PM	0.5455142
12	1	1	19	7:28:46 AM	9:24:36 PM	0.5804384
13	1	1	11	7:34:53 AM	4:18:36 PM	0.3636886
14	1	1	4	7:49:21 AM	10:08:46 PM	0.5968164
15	1	1	10	7:50:26 AM	6:51:44 PM	0.4592392
16	1	1	17	7:52:48 AM	12:28:25 PM	0.1914009
17	1	1	8	7:58:24 AM	9:24:30 PM	0.5597879
18	1	1	2	8:15:50 AM	1:55:41 AM	0.7360087
19	1	1	9	8:25:34 AM	9:28:50 PM	0.5439428
20	1	1	20	8:30:06 AM	8:53:55 PM	0.5165311
21	1	1	15	8:45:19 AM	9:07:54 PM	0.5156849
22	2	2	7	6:03:11 AM	10:54:41 AM	0.2024309
23	2	2	14	6:07:20 AM	9:32:08 AM	0.1422141
24	2	2	19	6:12:24 AM	8:22:18 PM	0.5902022
25	2	2	9	6:17:17 AM	11:56:48 AM	0.2357757
26	2	2	8	6:23:43 AM	8:34:28 PM	0.5907985
27	2	2	1	6:24:59 AM	10:28:53 AM	0.169377
28	2	2	4	6:38:12 AM	2:33:03 AM	0.8297565
29	2	2	13	6:45:23 AM	4:20:09 PM	0.3991435
30	2	2	16	6:57:04 AM	11:27:12 AM	0.1875978
31	2	2	12	7:00:24 AM	8:05:05 PM	0.5449194
32	2	2	17	7:07:10 AM	2:11:15 PM	0.2945043

Carpark Revenue Management

```
import pandas as pd
import math
from datetime import datetime
from datetime import timedelta
# Load re-generated time of arrival and exit for each hourly user
# Column names: Day, Weekday, ArrivalTime, ExitTime, LengthOfStay
hourly_data = pd.read_csv("C:\\Users\\Wei Zhang\\Box Sync\\Teaching\
\BA Program\\Quantitative Analysis Methods\\2019-2020\\Slides\
\Session_03_Data\\Hourly_ReGen.csv", header=0, index_col=0)
# Load re-generated time of exit and return for each monthly floating user
# Column names: Day, Weekday, CarNo, ExitTime, ReturnTime, Duration
monthly_data = pd.read_csv("C:\\Users\\Wei Zhang\\Box Sync\\Teaching\
\BA Program\\Quantitative Analysis Methods\\2019-2020\\Slides\
\Session_03_Data\\MonthlyFloating_ReGen.csv", header=0, index_col=0)

# Carpark parameter (Period measured by minute)
NS=30;NM=20;Period=15
# Initial state (monthly, hourly)
state={'nM':20,'nH':0}
# Define the trees related to different days of a week
weekday_tree={0:{'Time':datetime.strptime('12:0:0 AM','%I:%M:%S %p'),'Child':[],'State':state,
              'Type':'D','n':0,'V':0},'DayOfWeek':{1:1,2:1,3:1,4:1,5:1,6:0,7:0}}
weekend_tree={0:{'Time':datetime.strptime('12:0:0 AM','%I:%M:%S %p'),'Child':[],'State':state,
              'Type':'D','n':0,'V':0},'DayOfWeek':{1:0,2:0,3:0,4:0,5:0,6:1,7:1}}

def Expand(tree, node):
    if len(tree[node]['Child'])==0:
        tree[len(tree)-1]={'Time':tree[node]['Time'],'Child':[],'Parent':node,
                           'Decision':'Available','Type':'S','n':0,'V':0,'UCB':float('inf')}
        tree[len(tree)-1]={'Time':tree[node]['Time'],'Child':[],'Parent':node,
                           'Decision':'Full','Type':'S','n':0,'V':0,'UCB':float('inf')}
        tree[node]['Child']=[len(tree)-3,len(tree)-2]
```



Carpark Revenue Management

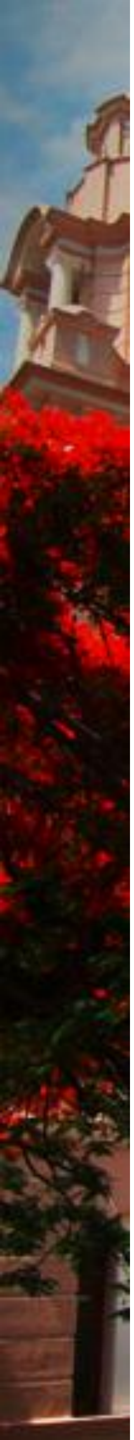
```
def BuildTree(tree, node, k):
    day=1
    for d in range(k):
        # Prepare a notebook to keep track of entrance and exit given any initial state
        notebook={'Die':0}
        time=tree[node]['Time']
        while time<datetime.strptime('11:59:59 PM', '%I:%M:%S %p'):
            notebook[time]={'Revenue':0,'Delta_nM':0,'Delta_nH':0}
            time=time+timedelta(minutes=Period)
        nM=tree[node]['State']['nM'];nH=tree[node]['State']['nH']
        # Initialize day and time for simulation
        while tree['DayOfWeek'][hourly_data['Weekday'][day].values[0]]==0:
            day=day+1
            if day>hourly_data.index[len(hourly_data.index)-1]:
                day=1
        total_H=len(hourly_data['Weekday'][day])
        index_H=0;index_M=0
        time=tree[node]['Time']
        # Start expansion, selection, and simulation
        pointer=node
        while time<=datetime.strptime('11:59:59 PM', '%I:%M:%S %p'):
            # Expanson
            if len(tree[pointer]['Child'])==0:
                Expand(tree, pointer)
            # Selection
            if tree[tree[pointer]['Child'][1]]['UCB']>tree[tree[pointer]['Child'][0]]['UCB']:
                pointer=tree[pointer]['Child'][1]
            else:
                pointer=tree[pointer]['Child'][0]
            # Simulation

            … …
```

Carpark Revenue Management

```
    # Simulation
    ... ...
    nM=tree[tree[pointer]['Parent']]['State']['nM']+notebook[time]['Delta_nM']
    nH=tree[tree[pointer]['Parent']]['State']['nH']+notebook[time]['Delta_nH']
    if nM+nH>NS:
        notebook['Die']=1
        break
    n_Child=len(tree[pointer]['Child'])
    while n_Child>0:
        if tree[tree[pointer]['Child'][n_Child-1]]['State']=={'nM':nM,'nH':nH}:
            pointer=tree[pointer]['Child'][n_Child-1]
            break
        else:
            n_Child=n_Child-1
    if n_Child==0:
        tree[len(tree)-1]={'Time':time+timedelta(minutes=Period),'Child':[],
                           'Parent':pointer,'State':{'nM':nM,'nH':nH},'Type':'D','n':0,'V':0}
        tree[pointer]['Child'].append(len(tree)-2)
        pointer=len(tree)-2
    time=tree[pointer]['Time']
# Start backpropagation
BSR=0
while pointer>node:
    pointer=tree[pointer]['Parent']
    BSR=BSR+notebook[tree[pointer]['Time']]['Revenue']*(1-notebook['Die'])
    tree[pointer]['V']=(tree[pointer]['V']*tree[pointer]['n']+BSR)/(tree[pointer]['n']+1)
    tree[pointer]['n']=tree[pointer]['n']+1
    if tree[pointer]['Type']=='S':
        tree[pointer]['UCB']=tree[pointer]['V']+2*(
            math.log(tree[tree[pointer]['Parent']]['n']+1)/tree[pointer]['n'])**0.5
```

Carpark Revenue Management

```
def Optimization(tree):
    node=0
    while tree[node]['Time']<datetime.strptime('12:0:0 AM','%I:%M:%S %p')+timedelta(days=1):
        print('Now is '+str(tree[node]['Time'].time())+'.')
        if tree[node]['n']==0:
            BuildTree(tree, node, 100)
        if tree[tree[node]['Child'][1]]['UCB']>tree[tree[node]['Child'][0]]['UCB']:
            node=tree[node]['Child'][1]
        else:
            node=tree[node]['Child'][0]
        print('For the next 15 min, show '+tree[node]['Decision']+'.')
        print('During this 15 min:')
        delta_nM=int(input('What is the net number of entrance of monthly users?'))
        delta_nH=int(input('What is the net number of entrance of hourly users?'))
        nM=tree[tree[node]['Parent']]['State']['nM']+delta_nM
        nH=tree[tree[node]['Parent']]['State']['nH']+delta_nH
        if nM+nH>NS:
            print('Game Over!')
            break
        else:
            n_Child=len(tree[node]['Child'])
            while n_Child>0:
                if tree[tree[node]['Child'][n_Child-1]]['State']=={'nM':nM,'nH':nH}:
                    node=tree[node]['Child'][n_Child-1]
                    break
                else:
                    n_Child=n_Child-1
            if n_Child==0:
                tree[len(tree)-1]={'Time':tree[node]['Time']+timedelta(minutes=Period),'Child':[],
                                   'Parent':node,'State':{'nM':nM,'nH':nH},'Type':'D','n':0,'V':0}
                tree[node]['Child'].append(len(tree)-2)
                node=len(tree)-2
```

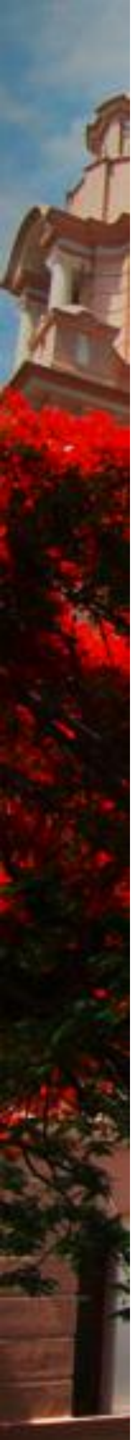


Carpark Revenue Management

The tree after 1000 days of simulation:

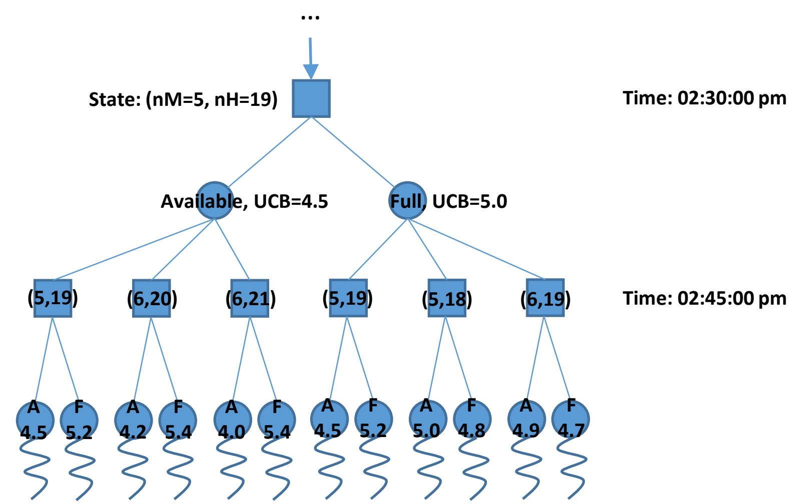
	A	B	C	D	E	F	G	H	I	J
1		Child	Decision	Parent	State	Time	Type	UCB	V	n
2	0	[1, 2]			{'nM': 20, 'nH': 0}	01-01-00 0:00	D		8.914706	1000
3	1	[3]	Available	0		01-01-00 0:00	S	9.149768	8.914706	500
4	2	[289]	Full	0		01-01-00 0:00	S	9.149785	8.914706	500
5	3	[4, 5]		1	{'nM': 20, 'nH': 0}	01-01-00 0:15	D		8.914706	500
6	4	[6]	Available	3		01-01-00 0:15	S	9.229986	8.914706	250
7	5	[25108]	Full	3		01-01-00 0:15	S	9.230037	8.914706	250
8	6	[7, 8]		4	{'nM': 20, 'nH': 0}	01-01-00 0:30	D		8.914706	250
9	7	[9]	Available	6		01-01-00 0:30	S	9.335047	8.914706	125
10	8	[49641]	Full	6		01-01-00 0:30	S	9.334894	8.914706	125
11	9	[10, 11]		7	{'nM': 20, 'nH': 0}	01-01-00 0:45	D		8.914706	125
12	10	[12]	Available	9		01-01-00 0:45	S	9.527275	8.857092	43
13	11	[90931]	Full	9		01-01-00 0:45	S	9.409195	8.944918	82
14	12	[13, 14]		10	{'nM': 20, 'nH': 0}	01-01-00 1:00	D		8.857092	43
15	13	[15]	Available	12		01-01-00 1:00	S	9.415678	9.415678	1
16	14	[168509]	Full	12		01-01-00 1:00	S	9.442298	8.843792	42
17	15	[16, 17]		13	{'nM': 20, 'nH': 0}	01-01-00 1:15	D		9.415678	1
18	16	[18]	Available	15		01-01-00 1:15	S	9.415678	9.415678	1
19	17	[]	Full	15		01-01-00 1:15	S	inf	0	0
20	18	[19, 20]		16	{'nM': 20, 'nH': 0}	01-01-00 1:30	D		9.415678	1
21	19	[21]	Available	18		01-01-00 1:30	S	9.415678	9.415678	1
22	20	[]	Full	18		01-01-00 1:30	S	inf	0	0
23	21	[22, 23]		19	{'nM': 20, 'nH': 0}	01-01-00 1:45	D		9.415678	1
24	22	[24]	Available	21		01-01-00 1:45	S	9.415678	9.415678	1
25	23	[]	Full	21		01-01-00 1:45	S	inf	0	0
26	24	[25, 26]		22	{'nM': 20, 'nH': 0}	01-01-00 2:00	D		9.415678	1
27	25	[27]	Available	24		01-01-00 2:00	S	9.415678	9.415678	1
28	26	[]	Full	24		01-01-00 2:00	S	inf	0	0

We need a really large amount of simulation to explore deeper in the tree!



In-Class Exercise

- Consider this tree built by the MCTS algorithm for the carpark problem. Now is 02:30:00 pm and there are 5 monthly users and 19 hourly users in the carpark. If for the next 15 minutes, there will be 2 hourly users arriving and 0 leaving. In addition, there will be 1 monthly floating users returning and 0 leaving. What sign should be shown after 15 minutes?



- When doing optimization, if we visit a state that has never been simulated before, we should:
  - A: Randomly pick an option
  - B: Run the MCTS algorithm in real time