CSE 250a Assignment 9.4

December 6, 2017

In [2]:
```python
import numpy as np
import pandas as pd
```

In [3]:
```python
#values
STATES = 81
ACTIONS = 4
GAMMA = 0.9925
ACTIONS_LIST = ['W','N','E','S']
MAZE = [3,11,12,15,16,17,
        20,22,23,24,26,29,
        30,31,34,35,39,43,
        48,52,53,56,57,58,
        59,60,61,62,66,70,71]

#load files
def parse_sparse(sparse_fh):
    sparse = np.loadtxt(sparse_fh)
    out_mtx = np.zeros([STATES,STATES])
    for row in sparse:
        out_mtx[int(row[1])-1,int(row[0])-1] = row[2]
    return out_mtx

# columns = s, rows = s' in transition mtxs
prob_a1 = parse_sparse('hw9_prob_a1.txt')
prob_a2 = parse_sparse('hw9_prob_a2.txt')
prob_a3 = parse_sparse('hw9_prob_a3.txt')
prob_a4 = parse_sparse('hw9_prob_a4.txt')
transition_mtxs = [prob_a1, prob_a2, prob_a3, prob_a4]
rewards = np.loadtxt('hw9_rewards.txt')
```

0.0.1 (a) Compute the optimal policy $\pi^*(s)$ and optimal value function $V^*(s)$ using *policy iteration*.

- $\pi'(s) = argmax_a[\sum_{s'} P(s'|s,a)V^\pi(s')]$
- $V^\pi = [I - \gamma P^\pi]^{-1}R$

In [4]:
```python
'''policy evaluation: calculate V(s) following pi (vec)
solve system of linear equations'''
```

```
def V_solve(pi):
    # construct nxn matrix of GAMMA*P(s'|s,pi(s))
    square = np.zeros([STATES,STATES])
    id_mtx = np.identity(STATES)
    for s in range(STATES): #s
        p_mtx = transition_mtxs[int(pi[s])]
        square[s,:] = id_mtx[s,:]-GAMMA*p_mtx[:,s]
    inv = np.linalg.inv(square) #invert square mtx
    V_s = np.dot(inv,rewards) #evaluate V(s) for all s
    return V_s

'''policy improvement: determine pi'(s)
s: current state
pi: current policy
return improved policy pi for state s'''
def policy_improvement(s, pi):
    vals = np.repeat(-np.inf, ACTIONS)
    for a in range(ACTIONS):
        vals[a] = np.sum(transition_mtxs[a][:,s]*V_solve(pi))
    return np.argmax(vals)

'''policy iteration to get optimal policy
return pi*, Vpi*'''
def policy_iteration():
    pi = np.random.randint(ACTIONS,size=STATES) #pi0 (random)
    Vpi = V_solve(pi) # corresponding to pi0
    while True:
        pi_new = np.repeat(1,STATES)
        for s in range(STATES):
            pi_new[s] = policy_improvement(s,pi)
        Vpi_new = V_solve(pi_new)
        if all(Vpi == Vpi_new):
            break
        pi = pi_new
        Vpi = Vpi_new
    return pi, Vpi

pi_opt, V_opt = policy_iteration()
```

(i) $V^*(s)$

In [5]:
```
Vopt_formatted = []
for val in V_opt:
    if val>0:
        Vopt_formatted.append(str(val))
    if val==0:
        Vopt_formatted.append('wall')
    if val<0:
```

```
        Vopt_formatted.append('dragon')
    pd.DataFrame(np.array(Vopt_formatted).reshape(9,9).T)
```

Out[5]:

	0	1	2	3	4
0	wall	wall	wall	wall	wall
1	wall	102.375264401	103.234623416	104.101212043	wall
2	100.700980727	101.523645149	wall	104.975075555	103.781407374
3	wall	wall	106.77826755	105.88853591	wall
4	wall	wall	107.674626429	wall	wall
5	wall	109.489934536	108.578487117	wall	wall
6	wall	110.409032962	wall	114.163229503	115.121557269
7	wall	111.335846634	112.270440318	113.212879322	wall
8	wall	wall	wall	wall	wall

	5	6	7	8
0	wall	wall	wall	wall
1	dragon	81.3994927813	dragon	wall
2	90.9853796009	93.6716558331	81.3994927813	wall
3	dragon	95.1728572646	dragon	wall
4	wall	108.342619343	wall	wall
5	dragon	109.583650718	dragon	wall
6	116.087929588	123.643070208	125.249789436	133.333333333
7	122.024912415	123.1822391	124.207385633	wall
8	wall	wall	wall	wall

(ii) $\pi^*(s)$

In [6]:
```
directions = [ACTIONS_LIST[each] for each in pi_opt]
print(np.array(directions).reshape(9,9).T)
```

```
[['W' 'W' 'W' 'W' 'W' 'W' 'W' 'W' 'W']
 ['W' 'E' 'E' 'S' 'W' 'W' 'S' 'W' 'W']
 ['E' 'N' 'W' 'S' 'W' 'W' 'S' 'W' 'W']
 ['W' 'W' 'S' 'W' 'W' 'W' 'S' 'W' 'W']
 ['W' 'W' 'S' 'W' 'W' 'W' 'S' 'W' 'W']
 ['W' 'S' 'W' 'W' 'W' 'W' 'S' 'W' 'W']
 ['W' 'S' 'W' 'E' 'E' 'E' 'E' 'E' 'W']
 ['W' 'E' 'E' 'N' 'W' 'E' 'E' 'N' 'W']
 ['W' 'W' 'W' 'W' 'W' 'W' 'W' 'W' 'W']]
```

0.0.2 (b) Value iteration

$V_{k+1}(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V_k(s')$

In [36]:
```
def value_iteration():
    Vk = np.repeat(0,STATES) #V0(s)=0 for all s
    Vk_new = np.repeat(-np.inf,STATES) #initialize Vk_new
```

```
    pi_opt = np.repeat(-np.inf,STATES) #initialize optimal policy pi*
    iter_count = 1
    while True:
        vals = np.full((ACTIONS,STATES),-np.inf)
        for a in range(ACTIONS):
            vals[a,:] = [np.sum(transition_mtxs[a][:,s]*Vk) for s in range(STATES)]
        Vk_new = np.array([rewards[s]+GAMMA*np.amax(vals[:,s]) for s in range(STATES)])
        if all(Vk_new == Vk):
            vals = np.full([ACTIONS,STATES],-np.inf)
            for a in range(ACTIONS):
                vals[a,:] = [np.sum(transition_mtxs[a][:,s]*Vk_new) for s in range(STAT
            pi_opt = np.array([np.argmax(vals[:,s]) for s in range(STATES)])
            break
        Vk = Vk_new
        iter_count += 1
    return Vk, pi_opt

Vopt2, pi_opt2 = value_iteration()
```

In [40]:
```
Vopt2_formatted = []
for val in Vopt2:
    if val>0:
        Vopt2_formatted.append(str(val))
    if val==0:
        Vopt2_formatted.append('wall')
    if val<0:
        Vopt2_formatted.append('dragon')
pd.DataFrame(np.array(Vopt2_formatted).reshape(9,9).T)
```

Out[40]:

	0	1	2	3	4 \
0	wall	wall	wall	wall	wall
1	wall	102.375264401	103.234623416	104.101212043	wall
2	100.700980727	101.523645149	wall	104.975075555	103.781407374
3	wall	wall	106.77826755	105.88853591	wall
4	wall	wall	107.674626429	wall	wall
5	wall	109.489934536	108.578487117	wall	wall
6	wall	110.409032962	wall	114.163229503	115.121557269
7	wall	111.335846634	112.270440318	113.212879322	wall
8	wall	wall	wall	wall	wall

	5	6	7	8
0	wall	wall	wall	wall
1	dragon	81.3994927813	dragon	wall
2	90.9853796009	93.6716558331	81.3994927813	wall
3	dragon	95.1728572646	dragon	wall
4	wall	108.342619343	wall	wall
5	dragon	109.583650718	dragon	wall
6	116.087929588	123.643070208	125.249789436	133.333333333

```
7  122.024912415    123.1822391  124.207385633           wall
8            wall           wall           wall           wall
```

```
In [42]: directions2 = [ACTIONS_LIST[each] for each in pi_opt2]
         print(np.array(directions2).reshape(9,9).T)
```

```
[['W' 'W' 'W' 'W' 'W' 'W' 'W' 'W' 'W']
 ['W' 'E' 'E' 'S' 'W' 'W' 'S' 'W' 'W']
 ['E' 'N' 'W' 'S' 'W' 'W' 'S' 'W' 'W']
 ['W' 'W' 'S' 'W' 'W' 'W' 'S' 'W' 'W']
 ['W' 'W' 'S' 'W' 'W' 'W' 'S' 'W' 'W']
 ['W' 'S' 'W' 'W' 'W' 'W' 'S' 'W' 'W']
 ['W' 'S' 'W' 'E' 'E' 'E' 'E' 'E' 'W']
 ['W' 'E' 'E' 'N' 'W' 'E' 'E' 'N' 'W']
 ['W' 'W' 'W' 'W' 'W' 'W' 'W' 'W' 'W']]
```

The values in the number squares in the maze agree with the results from part (b), where the values were obtained using policy iteration.