Stochastic Optimization

Edoardo Fantolino

Abstract

This document contains my ideas about the second exercise of the stochastic assignment. Here, I try to use dynamic programming (DP) and reinforcement learning (RL) to solve a production-maintenance problem.

This document is addressed to Professor Enrico Bibbona.

Keywords:dynamic programming, Markov decision problem, reinforcement learning, optimization.

1 Introduction

The main objective of this homework is to deal with a production-maintenance problem. We need to find the best strategy that will allow us to maximize our revenue. We will use dynamic programming techniques and simulations for reinforcement learning.

Our factory has 2 machines. They can be in three states:

- New N
- Worn W
- Broken B

In a working day, a new machine will create a revenue equal to 2 units. When a machine is worn, it will provide 1 unit of reward and when the machine is broken, it will not generate gain.

The first day the machines are new, but as time goes on, they can change state and became worn or even broken. We can choose two working methodology.

1. We will call the technician that will repair and set to new the machines only when both of them are broken. Every time that the technician is called we will pay 10 units.
2. We will call the technician more often. We have to pay 2 units just for the intervention, then 1.5 unit to repair worn machines and 4 unit for each broken machine.

Which is the best strategy?

2 Directed Graphs, Transition Probability Matrices and Transition Reward Matrices

(The images of the directed graph are at the end of the document because they were too huge.)

The directed graph represent the behaviour of our system. The nodes (circles) represent the different states, while the edges (arrows) are the possible trajectories that we can take. Near each edge there is a number that describe the probability of going from one state to another. We could describe this system with 6 or 9 states. We have 6 states when we don't distinguish the two machines while we have 9 states when we want to know which of the two machines is new, worn or broken. We will proceed with the 6 states case.

Working methodology number 1

The first working methodology tell us that we will call the technician only when the two machine are broken. You can see the Transition Probability Matrix (TMP_1) and Transition Reward Matrix (TRM_1) at the end of this page. We can observe that TPM is a stochastic matrix because the sum along the rows give always 1. In the $TMP_1 \in \mathbb{R}^{6\times6}$ we store the probability of going from one state to another, while the TRM_1 represent the gain or the loss that we are subject to when we make a certain step.

Working methodology number 2

The second working methodology tell to us that we can always call the technician. TPM_2 is a stochastic matrix while TRM_2 is the reward matrix. In this case TRM_2 is always a cost, so the entries of this matrix are negative. TPM_2 and TrM_2 are shown at the end of the page.

$$TPM_1 = \begin{array}{c} \\ NN \\ NW \\ NB \\ WW \\ WB \\ BB \end{array} \begin{array}{c} \begin{array}{cccccc} NN & NW & NB & WW & WB & BB \end{array} \\ \begin{pmatrix} 0.87\overline{1} & 0.12\overline{4} & 0 & 0.00\overline{4} & 0 & 0 \\ 0 & 0.87\overline{1} & 0.06\overline{2} & 0.06\overline{2} & 0.00\overline{4} & 0 \\ 0 & 0 & 0.9\overline{3} & 0 & 0.0\overline{6} & 0 \\ 0 & 0 & 0 & 0.87\overline{1} & 0.12\overline{4} & 0.00\overline{4} \\ 0 & 0 & 0 & 0 & 0.9\overline{3} & 0.0\overline{6} \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \qquad TRM_1 = \begin{array}{c} \\ NN \\ NW \\ NB \\ WW \\ WB \\ BB \end{array} \begin{array}{c} \begin{array}{cccccc} NN & NW & NB & WW & WB & BB \end{array} \\ \begin{pmatrix} 2 & 1.5 & 0 & 1 & 0 & 0 \\ 0 & 1.5 & 1 & 1 & 0.5 & 0 \\ 0 & 0 & 1 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 \\ -10 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$TPM_2 = \begin{array}{c} \\ NN \\ NW \\ NB \\ WW \\ WB \\ BB \end{array} \begin{array}{c} \begin{array}{cccccc} NN & NW & NB & WW & WB & BB \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \qquad TRM_2 = \begin{array}{c} \\ NN \\ NW \\ NB \\ WW \\ WB \\ BB \end{array} \begin{array}{c} \begin{array}{cccccc} NN & NW & NB & WW & WB & BB \end{array} \\ \begin{pmatrix} -2 & 0 & 0 & 0 & 0 & 0 \\ -3.5 & 0 & 0 & 0 & 0 & 0 \\ -6 & 0 & 0 & 0 & 0 & 0 \\ -5 & 0 & 0 & 0 & 0 & 0 \\ -7.5 & 0 & 0 & 0 & 0 & 0 \\ -10 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

3 Q-factors value iteration algorithm

The key equation of the algorithm is the following one:

$$Q^{k+1}(i,a) = \sum_{j \in S} p_a(i,j) \left[r(i,a,j) + \lambda \max_{b \in A} Q^k(j,b) \right]$$

The Q matrix contain the Q-factors. The value function is obtain as follow:

$$J^*(i) = \max_{a \in A} Q(i,a)$$

As you can see from Tab.1, the algorithm suggest us to call the technician as soon as a machine is worn. With this strategy we will only visit the following states: NN, NW, WW. Probably, the algorithm choose this strategy because we have a low probability of damaging the machine, and if it happens it is better to repair it fast in order to come back to the state where there is the highest reward.

To see if the results given by the algorithm made sense I run several simulations with different fixed strategies that allowed me to compute the reward.

- Call the technician only when the two machines are broken.
  Total discounted reward linked to policy (1,1,1,1,1,1) $\approx$ 165.82
- Call the technician when one or two machines are broken.
  Total discounted reward linked to policy (1,1,2,1,2,1) $\approx$ 226.87
- Call the technician as soon as a machine is worn.
  Total discounted reward linked to policy (1,2,2,2,2,1) $\approx$ 260.44

I also tried to change the technician costs in order to see if the result changed as well. For example I tried to make inconvenient to repair the worn machine going from cost of repair equal to 1.5 up to 3.5. In this case, the algorithm reply by favouring in the worn states the strategy 1 as expected.

$state$	$Q_{action=1}$	$Q_{action=2}$	J	μ
NN	261.9722	258.6623	261.9722	1
NW	257.0445	257.1623	257.1623	2
NB	254.2562	254.6623	254.6623	2
WW	254.9857	255.6623	255.6623	2
WB	252.1973	253.1623	253.1623	2
BB	250.6623	250.6623	250.6623	1-2

TABLE 1: The results obtained with the Q-factors value iteration algorithm

4 Reinforcement Learning

The key equation for the Q-learning algorithm is the following:

$$Q^{k+1}(i,a) \longleftarrow (1-\alpha_{k+1})Q^k(i,a) + \alpha_{k+1}\left[r(i,a,j) + \lambda \max_{b\in A} Q^k(j,b)\right]$$

The algorithm use the main concepts of DP Q-factors value iteration algorithm, but it is putted in a context of simulations. The call it learning because it will update the Q-factors value thanks to a simulation. So as the simulation goes on, the Q-learning algorithm will approximate better and better the Q-factors, and so we can make an analogy with the human idea of learning (trials-errors).

4.1 The results of Q-learning and Q-factors value iteration

We can compare Tab.1 and Tab.2. We see that some of the values obtained with Q-learning approach the ones found by the Q-factor value iteration while other are not quite near. To understand why this phenomenon happen, it is useful to see how many times the simulation pass through a given state. Thanks to Tab.3 it is clear that we have a better approximation for the state that we visit more times, while for the states that we visit fewer times the approximation is worse. This is due to the "update behaviour" of the algorithm.

In Tab.3 we see that in state NN we jump more that 88 thousands times so the algorithm can approximate this values with a good precision, but this is not the case for state BB, where the algorithm pass less that 100 times. I realized some graph in order to see how much this approximation is influenced by the number of iteration. I repeated the RL algorithm 10 times in order to find some reliable data. I computed the standard deviation and the mean of the distribution of the data. You can see the results in Fig.3.

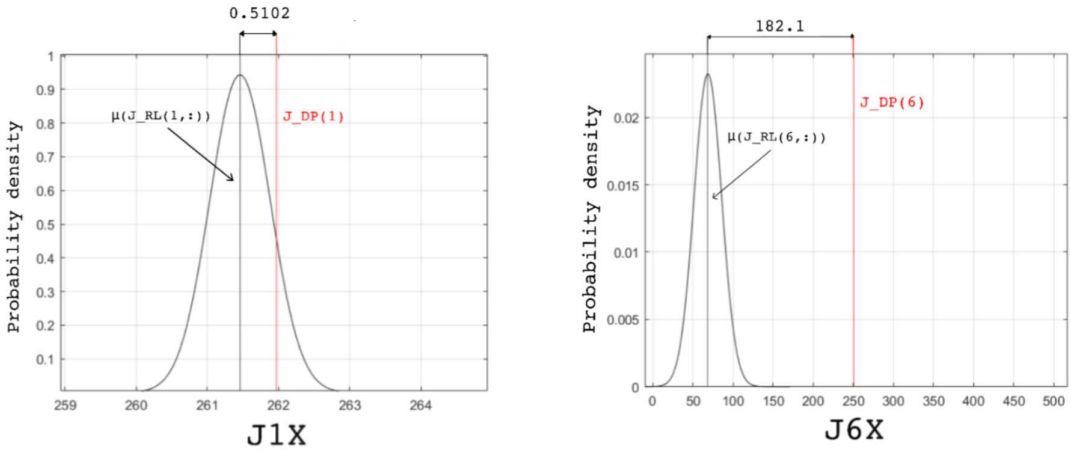
Fig.1 tell us that we will have a low error in the approximation of the Q-factors of the state where we pass many times, while for the state that we visit few times we will have a bigger error. In fact, the gap between the mean of the distribution and the actual value of the Q-factor found with DP increase when the number of visit decrease.

$state$	$Q_{action=1}$	$Q_{action=2}$	J	μ
NN	260.985	257.749	260.985	1
NW	256.184	256.288	256.288	2
NB	253.253	253.760	253.760	2
WW	253.233	254.764	254.764	2
WB	239.275	251.570	251.570	2
BB	55.2059	77.2890	77.2890	2

TABLE 2: The results obtained with the RL algorithm

state	number of visits
NN	885626
NW	98104
NB	5749
WW	8659
WB	1788
BB	74

TABLE 3: number of visits per state



(A) Distribution RL state NN (B) Distribution RL state BB

FIGURE 1: Distribution probability for state NN and state BB.

4.2 Reinforcement Learning with greedy action choice

I started thinking about how the algorithm learn and how it choose which action to perform. As stated in the slides and in the book, the standard algorithm choose the action uniformly at random from the set of the possible actions. So I implemented a more advanced algorithm that improve the choice of the action because it take into account what it has learned till that simulation. The main difference is that we will choose action a in state i with probability p_k.

$$a = \arg\max_{b \in A(i)} Q(i,b) \quad and \quad p_k = 1 - \frac{B^k}{\ln k}$$

As this is not part of the assignment I put the results at the end of the document.

5 Conclusion

I think that given the results shown above, the algorithms behave as one should expect. The best choice in this case is to repair the machines as soon as possible. If I have to find a reason I would like to make an analogy. Think about the cars in formula 1. If a pilot has a hole in a wheel (worn wheel) or if the wheel is broken, the team make the pilot do a pit stop as soon as possible. They don't wait many laps to change the wheel, because it is better to repair the damage and then let the car go as fast as it can.

References

Abhijit Gosavi. Similation-Based Optimization, Parametric Optimization Techniques and Reinforcement Learning. Springer (2003).

Bibbona Enrico, DISMA, Politecnico di Torino. Stoch_Opt_9.pdf (2020-2021).

Code

Initialization of the parameter:

```
close all
clear
clc
p12 = 1/15;
p11 = 1-p12;
p23 = 1/15;
p22 = 1-p23;

P1 = [(1-2*(p12*p11)-(p12*p12)) (p12*p11)+(p12*p11) 0 (p12*p12) 0 0 ;
        0 (1-2*(p12*p11)-(p12*p12)) (p23*p11) (p22*p12) (p12*p23) 0 ;
        0 0 (p11) 0 (p12) 0;
        0 0 0 (1-2*(p22*p23)-(p23*p23)) (p23*p22)+(p23*p22) (p23*p23);
        0 0 0 0 (p22) (p23);
        1 0 0 0 0 0];

P2 = [1 0 0 0 0 0;
      1 0 0 0 0 0;
      1 0 0 0 0 0;
      1 0 0 0 0 0;
      1 0 0 0 0 0;
      1 0 0 0 0 0];

R1 = [2 1.5 0 1 0 0;
      0 1.5 1 1 0.5 0;
      0 0 1 0 0.5 0;
      0 0 0 1 0.5 0;
      0 0 0 0 0.5 0;
      -10 0 0 0 0 0];


R2 = [-2 0 0 0 0 0;
      -3.5 0 0 0 0 0;
      -6 0 0 0 0 0;
      -5 0 0 0 0 0;
      -7.5 0 0 0 0 0;
      -10 0 0 0 0 0];

P = P1;
P(:,:,2) = P2;
R = R1;
R(:,:,2) = R2;
```

Q-factor algorithm

Here there is the code of the Q-factor algorithm

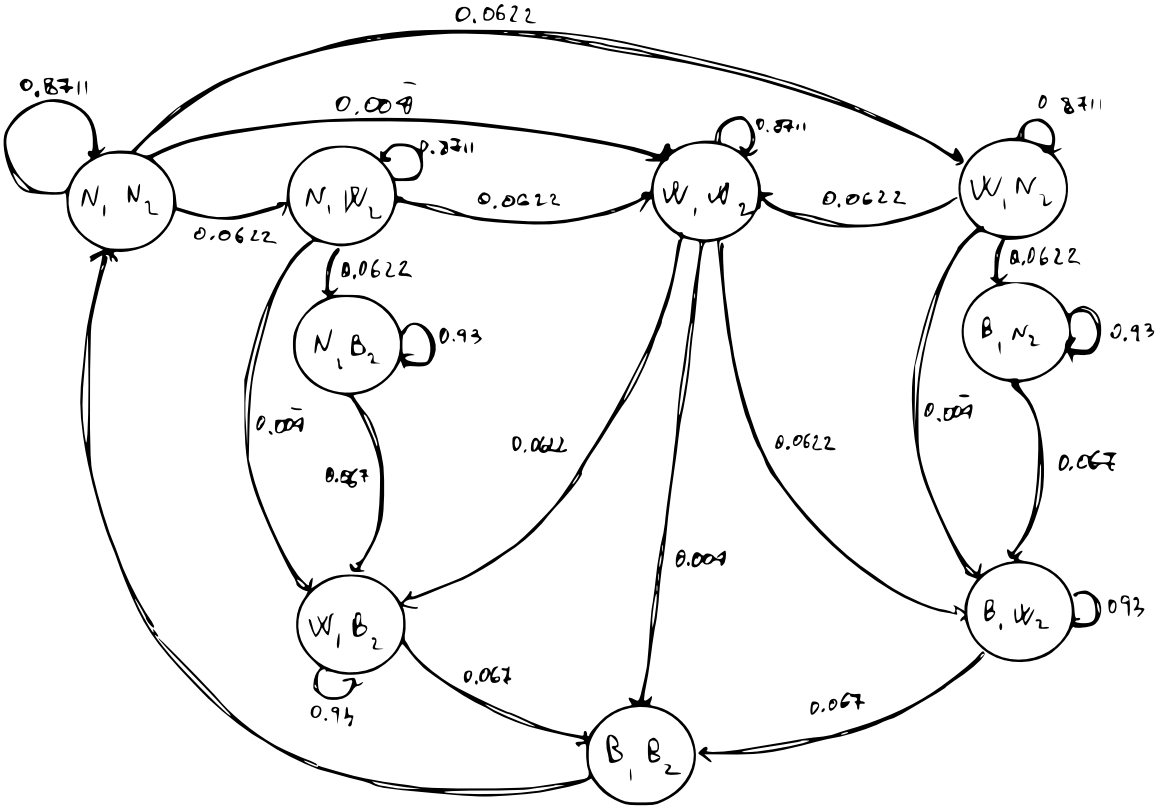
```
sizes = size(P);
number_of_actions = sizes(3);
number_of_states = sizes(1);
k = 1;
J = zeros(number_of_states,1);
Jn = zeros(number_of_states, 1);
Q = zeros(number_of_states, number_of_actions);
epsilon = 0.001;
sp = 2*epsilon;
lambda = 0.995;

while sp > (((epsilon*(1-lambda))/(2*lambda)))
% for i=1:1e4
    for z=1:number_of_states

        partial = zeros(number_of_states, 1);
        for j=1:number_of_actions
            rbar = P(z,:,j)*R(z,:,j)';
            summation = lambda * sum(P(z,:,j)*J(:));

            partial(j) = rbar + summation;
            Q(z,j) = partial(j);
        end
        Jn(z) = max(partial);

    end

    sp = norm(Jn-J);
    J = Jn;
end


for i=1:number_of_states
    [argvalue,argmax] = max(Q(i,:));
    mu(i) = argmax;
end
```

Q-learning algorithm

Here there is the code of the Q-learning algorithm

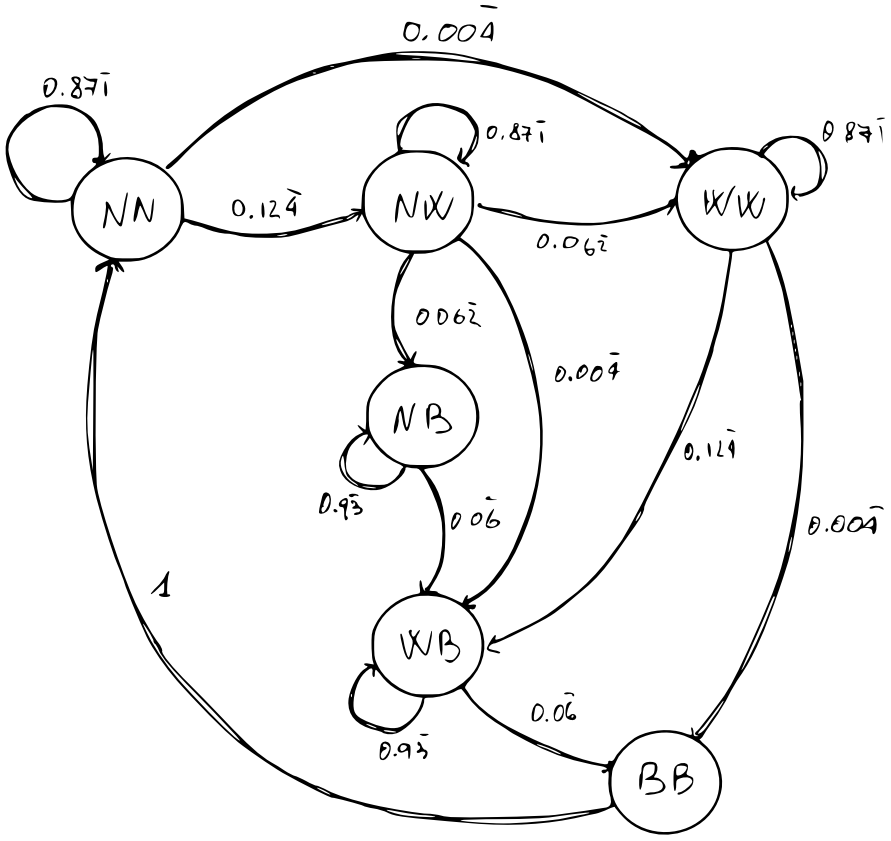
```
number_of_actions = 2;
number_of_states = length(P);
k = 1;
J = zeros(number_of_states,1);
Jn = zeros(number_of_states, 1);
Q = zeros(number_of_states, number_of_actions);
lambda = 0.995;
A = 1500;
B = 3000;
max_number_of_iterations = 1e6;
states = zeros(max_number_of_iterations,1);
actions = zeros(max_number_of_iterations,1);

states(1) = 1;
actions(1) = 1;
Q=zeros(number_of_states,2);

for k=2:max_number_of_iterations

    alpha = A/(B+k);
    actions(k) = randi(number_of_actions);
    states(k) = find(mnrnd(1,P(states(k-1),:,actions(k))));

    r = R(states(k-1), states(k), actions(k));
    q = max(Q(states(k),:));
    Q(states(k-1), actions(k)) = ...
            (1-alpha)*Q(states(k-1),actions(k))...
            + alpha*(r+lambda*q);
end

for i=1:number_of_states
    [argvalue,argmax] = max(Q(i,:));
    Jn(i) = max(Q(i,:));
    mu(i) = argmax;
end

mu
Jn
Q
[GC,GR] = groupcounts(states);
```

Appendix



$$
TPM = \begin{array}{c} \\ N_1N_2 \\ N_1W_2 \\ W_1N_2 \\ N_1B_2 \\ B_1N_2 \\ W_1W_2 \\ W_1B_2 \\ B_1W_2 \\ B_1B_2 \end{array}
\begin{array}{c}
\begin{array}{ccccccccc} N_1N_2 & N_1W_2 & W_1N_2 & N_1B_2 & B_1N_2 & W_1W_2 & W_1B_2 & B_1W_2 & B_1B_2 \end{array} \\
\begin{pmatrix}
0.87\overline{1} & 0.06\overline{2} & 0.06\overline{2} & 0 & 0 & 0.00\overline{4} & 0 & 0 & 0 \\
0 & 0.87\overline{1} & 0 & 0.06\overline{2} & 0 & 0.06\overline{2} & 0.00\overline{4} & 0 & 0 \\
0 & 0 & 0.87\overline{1} & 0 & 0.06\overline{2} & 0.06\overline{2} & 0 & 0.00\overline{4} & 0 \\
0 & 0 & 0 & 0.9\overline{3} & 0 & 0 & 0.0\overline{6} & 0 & 0 \\
0 & 0 & 0 & 0 & 0.9\overline{3} & 0 & 0 & 0.0\overline{6} & 0 \\
0 & 0 & 0 & 0 & 0 & 0.87\overline{1} & 0.06\overline{2} & 0.06\overline{2} & 0.00\overline{4} \\
0 & 0 & 0 & 0 & 0 & 0 & 0.9\overline{3} & 0 & 0.0\overline{6} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.9\overline{3} & 0.0\overline{6} \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\end{array}
$$

FIGURE 2: Directed graph and TPM distinguishing between the two machines and working methodology 1.



$$TPM_1 = \begin{array}{c} \\ NN \\ NW \\ NB \\ WW \\ WB \\ BB \end{array} \begin{array}{c} \begin{array}{cccccc} NN & NW & NB & WW & WB & BB \end{array} \\ \begin{pmatrix} 0.87\overline{1} & 0.12\overline{4} & 0 & 0.00\overline{4} & 0 & 0 \\ 0 & 0.87\overline{1} & 0.06\overline{2} & 0.06\overline{2} & 0.00\overline{4} & 0 \\ 0 & 0 & 0.9\overline{3} & 0 & 0.0\overline{6} & 0 \\ 0 & 0 & 0 & 0.87\overline{1} & 0.12\overline{4} & 0.00\overline{4} \\ 0 & 0 & 0 & 0 & 0.9\overline{3} & 0.0\overline{6} \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$TRM_1 = \begin{array}{c} \\ NN \\ NW \\ NB \\ WW \\ WB \\ BB \end{array} \begin{array}{c} \begin{array}{cccccc} NN & NW & NB & WW & WB & BB \end{array} \\ \begin{pmatrix} 2 & 1.5 & 0 & 1 & 0 & 0 \\ 0 & 1.5 & 1 & 1 & 0.5 & 0 \\ 0 & 0 & 1 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 \\ -10 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

FIGURE 3: Directed graph, TPM and TRM for the more general approach. The use of working methodology 1 is made.

Results of Reinforcement Learning with greedy action choice

$state$	$Q_{action=1}$	$Q_{action=2}$	J	μ
NN	261.283	258.278	261.283	1
NW	254.399	256.288	256.678	2
NB	27.8090	232.573	232.573	2
WW	177.057	255.251	255.251	2
WB	1.75900	105.598	105.598	2
BB	22.6053	0	22.6053	1

TABLE 4: The results obtained with the RL greedy algorithm

Here you can see the result that I obtained with the Reinforcement Learning algorithm with a different strategy to choose the action.

The code is the same as the standard Q-learning but we have to change:

```
actions(k) = randi(number_of_actions);
```

in this:

```
if rand <= 1-Bk/log(k)
  [argvalue,argmax] = max(Q(states(k-1),:));
  actions(k) = argmax;
else
  [argvalue,argmin] = min(Q(states(k-1),:));
  actions(k) = argmin;
end
```

state	**number of visits**
NN	885402
NW	110025
NB	352
WW	4156
WB	63
BB	2

TABLE 5: number of visits per state RL greedy