Elevator group control using reinforcement learning

Bo Fan

We define the following instances:

Instance1: number of floors= 10, number of elevators=2. (**elevatorsinstmdp_150hw3.rddl**)

Instance2: number of floors=20, number of elevators=4. (**elevatorsinstmdp_150hw3L.rddl**)

1 Results of the Random Concurrent Policy (RandomPolicy.java)

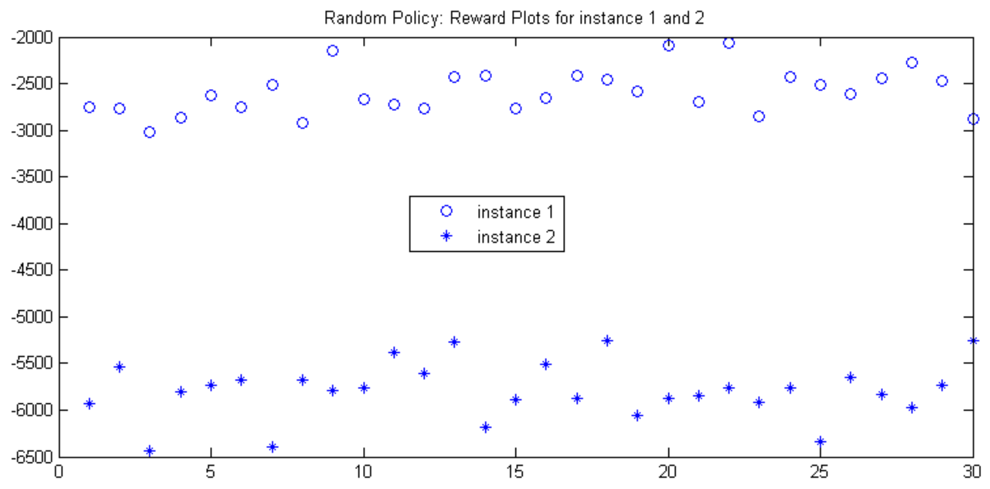


Fig 1 Random Policy: mean value plots

Figure1 and the following table separately show the reward values as a function of runs and the corresponding mean value and standard deviation of the rewards.

	Instance 1 Random Policy	Instance 2 Random Policy
Mean	-2585.7	-5790.3
Std	243.6	301.4

2 Results of the Round Robin Policy (RRPolicy.java)

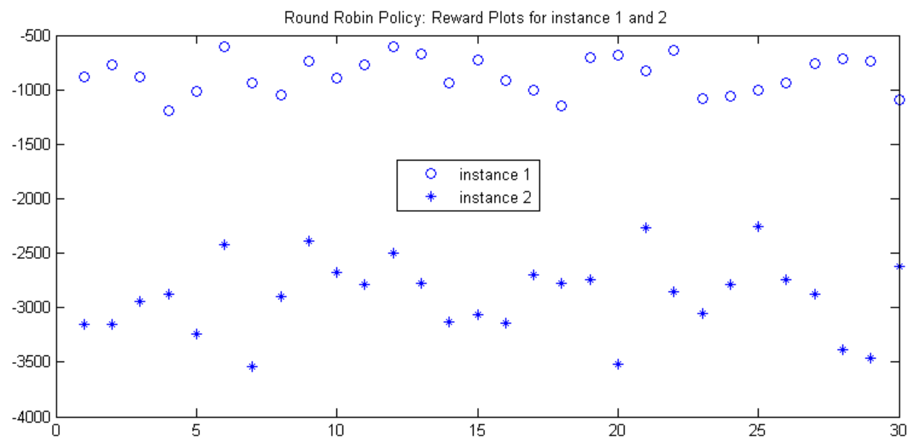


Fig 2 Round Robin Policy: mean value plots

Figure 2 shows the reward values as a function of runs. The following table shows the corresponding mean value and the standard deviation of the rewards.

	Instance 1 Round Robin	Instance 2 Round Robin
Mean	-866.1	-2890.3
Std	169.1	347.4

3 Results of My Policy (MPolicy.java)

3.1 The idea of my policy

My policy works better for even number of elevators which is written according to the round robin policy. We keep all of the assumptions of Round Robin and add two circumstances. Those are the situations where Round robin policy does not concern about.

1) My constraint: Open-door-* action case will not happen for two elevators at the same floor simultaneously.

When the number of elevators is greater than 2, there is a high probability that two elevators will be open-door-go - up/down with the same direction of up/down value at the same floor. To avoid this, we develop our constraints to judge whether the different elevators will have the same open-door-go-up/down action. If that happens, the next elevator will be given the action of go-cur-direction.

2) Two new floors where elevators can change the directions

Besides top and bottom floor, we add two floors (F4,F5) where the elevators can change the directions.

Case 1	Case 2
F9	F9
F8	F8 person wait up
F7 person wait up	F7 person wait up
F6	F6
F5	At F5=Upper position cur-dir-down
At F4=Lower position cur-dir-up	F4
F3	F3
F2 person wait down	F2 person wait down

F1 person wait down	F1
F0	F0

We take instance 1 as an example. Floor 4 (F4) is called the lower position, and F5 is called the upper position. In case 1 from the table, when the elevator is at F4 with cur-direction going up, nobody in the elevator will go up, and the total number of person-go-up from the upper half floors (F5-F9) is less than the total number of person-go-down from the lower half floors (F0-F4). It will change its direction to go-down (number of wait-up==1 < number of wait-down==2 in case1). Similarly, when the elevator is at F5 with current direction going down, nobody in the elevator will go down and the total number of person-go-up from the upper half floors (F5-F9) is greater than the total number of person-go-down from the lower half floors (F0-F4). It will change its direction to go-up (number of wait-up==2 > number of wait-down==1 in case2).

3.2 results

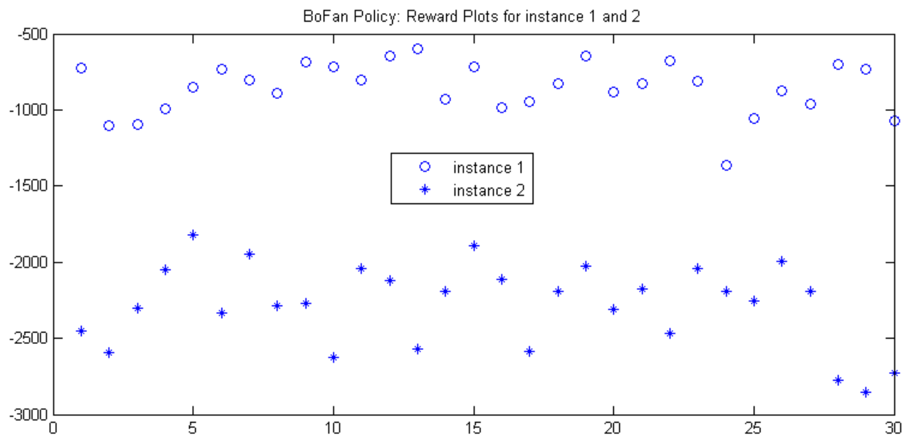


Fig 3 My policy: mean value plots

Figure 3 shows the reward values as a function of runs. The following table shows the corresponding mean value and standard deviation.

	Instance 1 MPolicy	Instance 2 MPolicy
Mean	-855.4	-2279.7
Std	172.1	270.2

4 Comparison of the results from two instances

In all of the experiment, we assume nobody will wait at the top or the bottom floor. We also find that if the number of floors is not changed, instance with more elevators will have larger rewards. It is because more elevators will reduce the waiting time. If the number of elevators is not changed, instance with fewer floors will have bigger rewards.

In the following table, we can see that our method (MPolicy) outperforms the random policy and the round robin policy in the mean value. Besides, Round robin is better than random policy. Especially for instance 2, our method is 600 larger than round robin in rewards. For the same policy, instance with more floors and elevators will give larger STD and smaller mean value. The rewards of the three methods are plotted together in fig 4 and 5. In fig 5, the reward of our method is bigger than the other two methods nearly for each run.

	Instance 1 Random Policy	Instance 2 Random Policy	Instance 1 Round Robin	Instance 2 Round Robin	Instance 1 MPolicy	Instance 2 MPolicy
Mean rewards	-2585.7	-5790.3	-866.1	-2890.3	-855.4	-2279.7
Std rewards	243.6	301.4	169.1	347.4	172.1	270.2

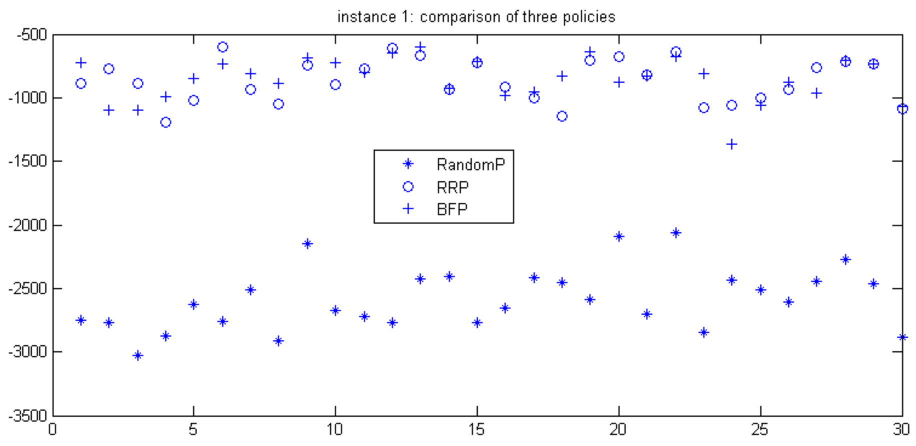


Fig 4 Three methods: mean value plots in instance 1

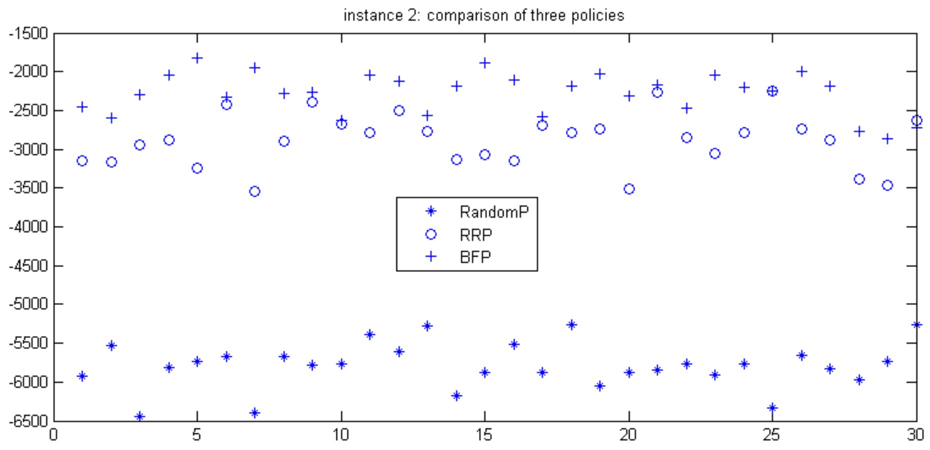


Fig 5 Three methods: mean value plots in instance 2

5 Results from Rollout policy when pi is random (RolloutRandomPolicy.java)

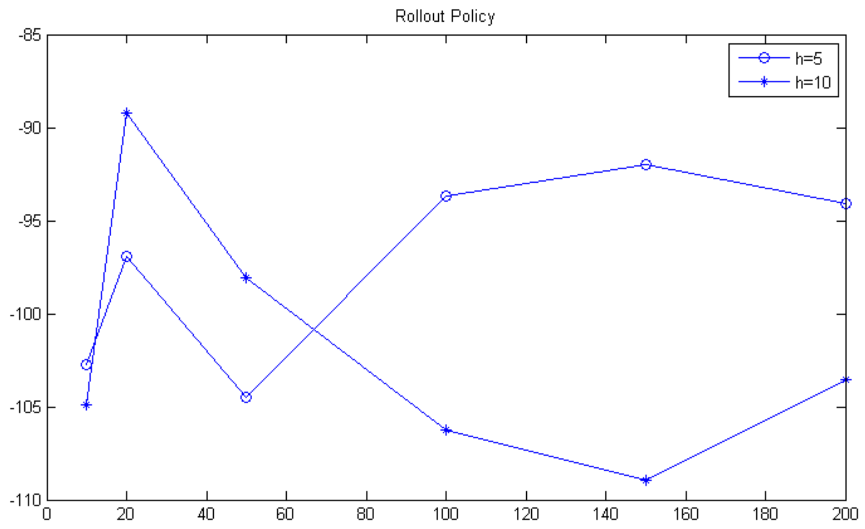
5.1 Mean and standard deviation

Because the action is chosen randomly, we use a legal action criterion to determine whether a random action is reasonable or not. For example, when the door is closed, the next action should not be close-door. The mean table is shown below (W is a parameter of the algorithm controlling the number of samples):

Mean	W=10	W=20	W=50	W=100	W=150	W=200
H=5	-102.7250	-96.9250	-104.4750	-93.6750	-92	-94.0750
H=10	-104.9250	-89.2000	-98.1000	-106.2750	-108.9500	-103.5750

The standard deviation table is shown below:

Std	W=10	W=20	W=50	W=100	W=150	W=200
H=5	20.5289	16.8630	15.4369	18.9795	23.7139	17.4992
H=10	19.1946	15.7117	20.6270	23.1335	16.3918	19.6621

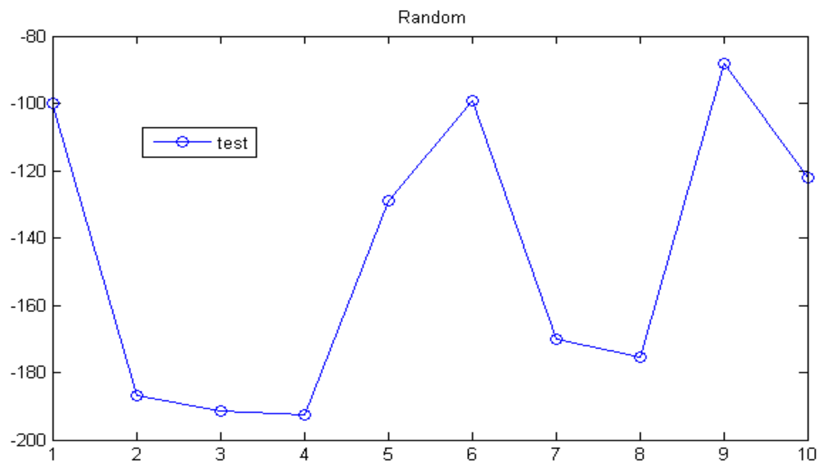


It can be seen in the figure that, when w >50, the performance of h=5 is much better than h=10. For w<50, h=10 is better. When h=10, the reward function of w is nearly increasing.

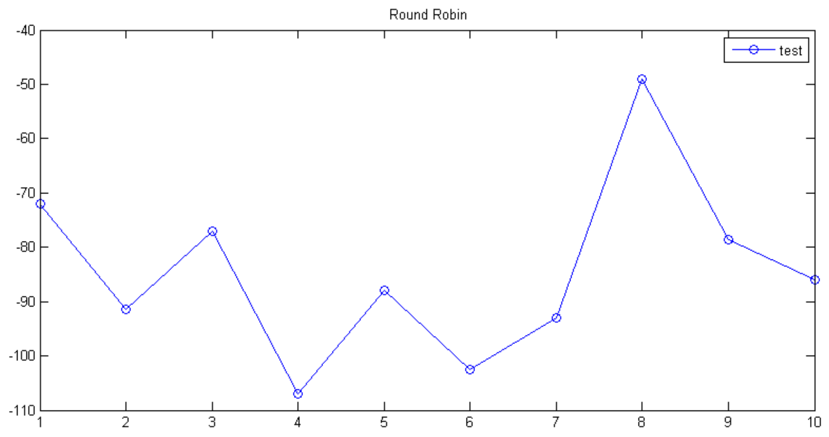
5.2 Comparison to the other methods

	Random concurrent	Round robin
mean	-145.4750	-84.4500
Std	42.0126	16.5822

From this table we can find that round robin is better than rollout policy with pi=random policy, and rollout policy method is better than random concurrent method. The following two figures show the results for random policy and round robin separately.



Rewards of random concurrent policy as a function of episode.



Rewards of round robin policy as a function of episode.

6 Simulation results from Rollout policy pi=Round robin (RolloutRRPolicy.java)

6.1 Mean values and standard deviation

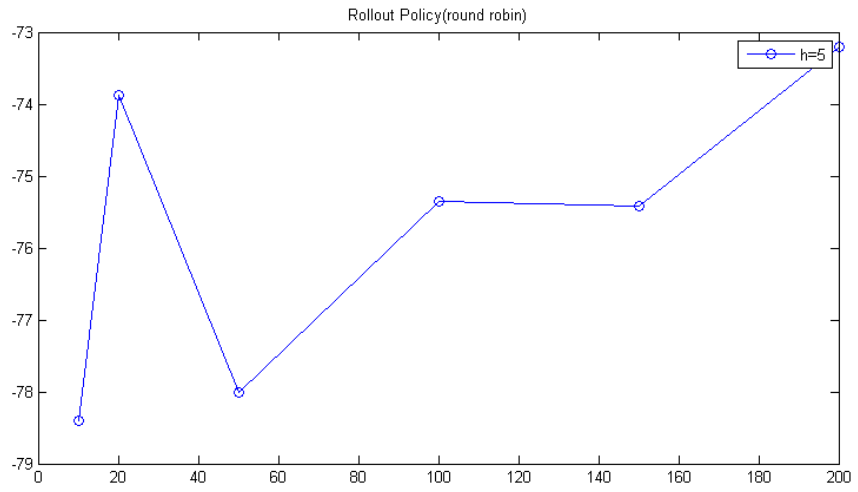
The mean average table is shown below:

Mean	W=10	W=20	W=50	W=100	W=150	W=200
H=5	-78.4	-73.8750	-78	-75.3500	-75.4250	-73.2000

The mean standard deviation table is shown below:

Std	W=10	W=20	W=50	W=100	W=150	W=200
H=5	19.2214	20.1192	24.2899	15.8549	25.4619	26.3069

6.2 Plots

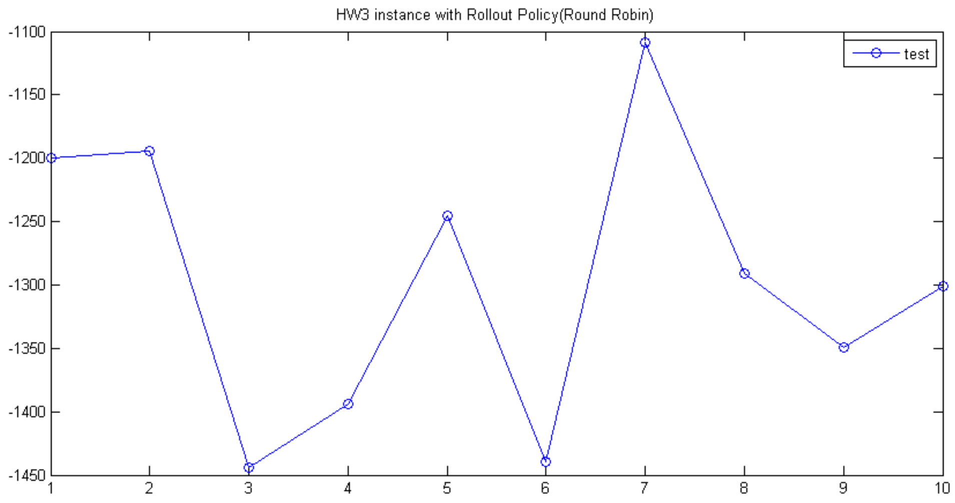


We only plot the reward as a function of w when h=5. When w=200, the reward function arrives at its maximum value.

7 Results for hw3 using Rollout policy with pi=round robin policy

When H=10,W=50, the mean reward value is -1296.7. The standard deviation is 111.7068.

The following figure shows the reward as a function of episode.



8 Results of cross traffic instance from rollout policy with pi=random policy (RolloutTrafficPolicy2.java)

In this domain, we have 4 actions for a robot. It can go to four directions. The action is -38.6 and the std is 4.427. I do not write the action criterion function in this domain with java, but I put the pseudo code as follows:

```
public boolean checkreasonable(State cs,ArrayList<PVAR_INST_DEF> factions ){
    If (robot at [x1,y1], go-N,go-W){ Return false;}
    Else if (robot at [x4,y1], go-N,go-W) ){ Return false;}
    Else if (robot at [x1,y4], go-N,go-W) ){ Return false;}
    Else if(robot at [x4,y4], go-N,go-W) ){ Return false;}
    Else if(robot at [x2,y1], go-N) ){ Return false;}
    Else if(robot at [x3,y1], go-N) ){ Return false;}
    Else if(robot at [x1,y2], go- W) ){ Return false;}
    Else if(robot at [x1,y3], go-W) ){ Return false;}
    Else if(robot at [x4,y2], go-E) ){ Return false;}
    Else if(robot at [x4,y3], go-E) ){ Return false;}
    Else if(robot at [x2,y4], go-S) ){ Return false;}
    Else if(robot at [x3,y4], go-S) ){ Return false;}
    Else {return true;}
}
```