گزارش تمرین شماره ۲

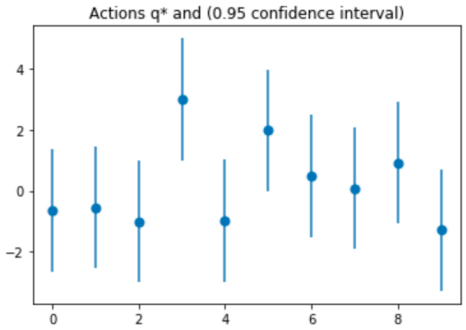
نام و نام خانوادگی	علی الهی
شماره دانشجویی	۸۱۰۶۹۶۳۳۶

حل سوالات این تکلیف در فایل های RL-HW#۲-Q#۱، RL-HW#۲-Q#۲ و RL-HW#۲-Q#۳ در و فرمت ipynb و html قرار دارند. درکد ها، مراحل کلی حل مسئله شامل تعریف کلاس‌ها، پیاده‌سازی الگوریتم، بررسی نتایج و tune کردن hyperparameter ها با markdow مشخص شده.

در تمامی سوالات، hyperparameter ها از جمله e در الگوریتم e-greedy و learning rate ها یا با آزمون و خطا و یا با استفاده از cross-validation، tune شده اند اما به دلیل کاهش حجم کد این مشاهدات حذف شده ان اما نتایج قابل مشاهده هستند.

سوال اول

در این سوال ابتدا مطابق شکل ۱-۲ کتاب ساتون-بارتو با یک مسئله 10-armed-badit مواجه هستیم. هر arm میانگین q_{a_i} و واریانس یک دارد که q_{a_i} ها خود توضیعی با میانگین صفر و واریانس ۱ می آیند.



برای محاسبه optimal action در این مسئله از الگوریتم Thompson sampling استفاده می کنیم. همچنین طبق Bayesian belief revision باور هایی از میانگین و واریانس های action ها داریم و طی هر مشاهده آنها را آپدیت می کنیم. شبه کد یافتن بهترین action در مشاهده iام و update کردن باور ها در ادامه آمده است. (میانگین و واریانس باورها توسط μ_k و π_k نشان داده شده اند.)

While true:

for k in [1, …, umber of actions]:

θ_k = sample from kth action

A_t = Argmax θ_k

do A_t, get R_t

update μ_k and π_k using bayesian belief revision:

$\mu_k = \mu_k$ + learnig_rate (R_t - μ_k)

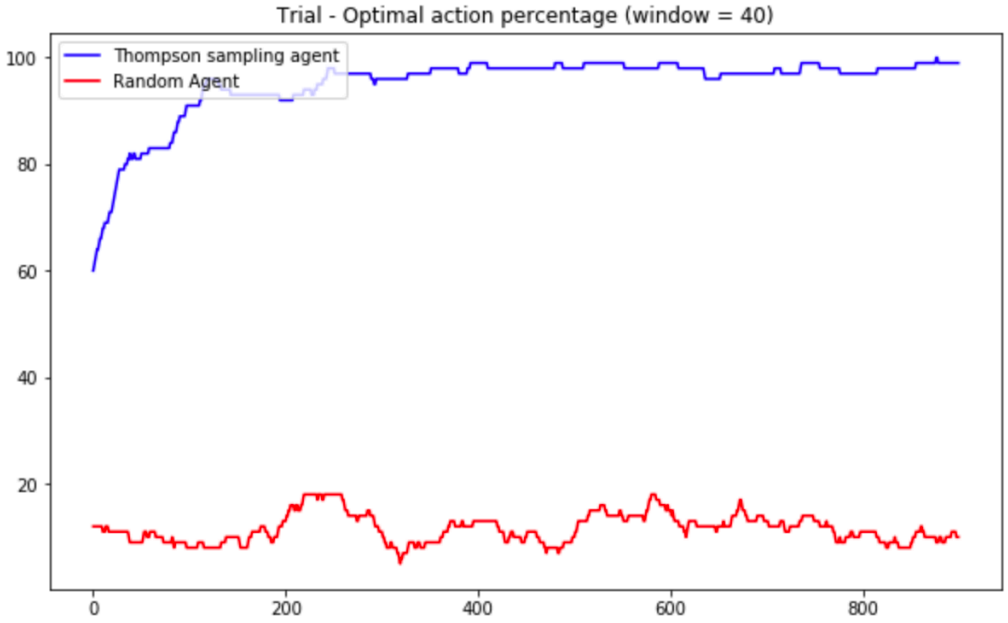
$\pi_k = \pi_k + \pi_{observation}$

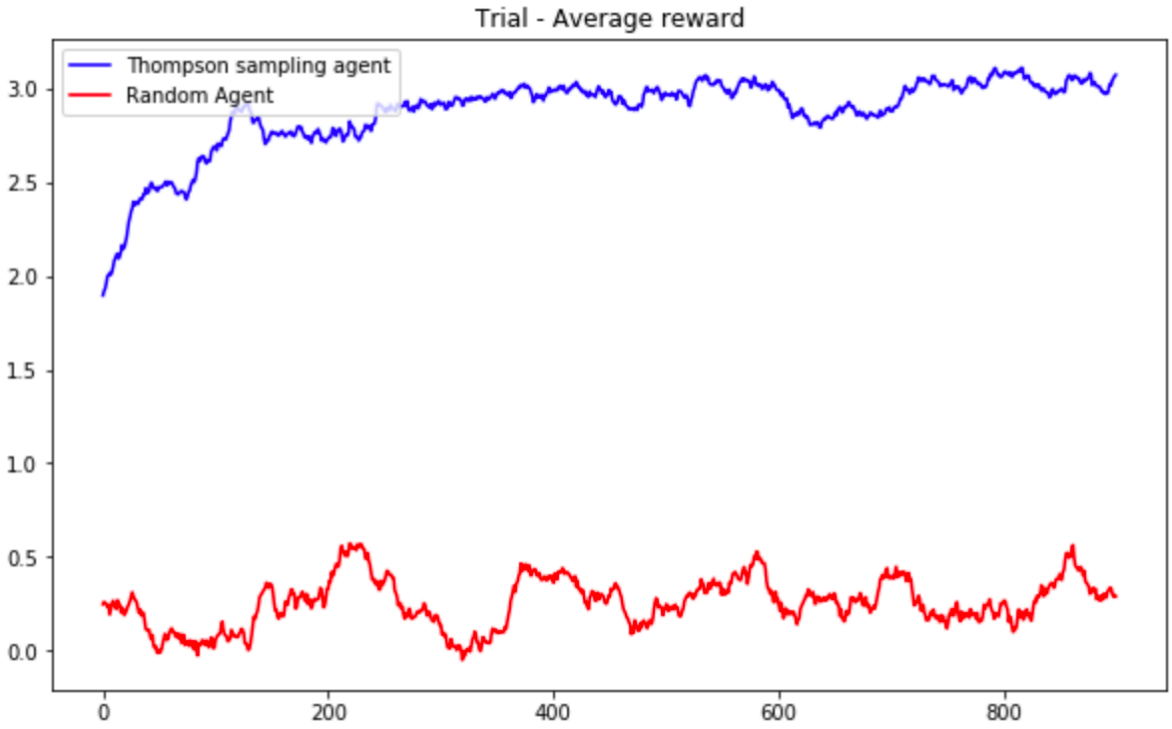
در این مسئله دو hypeparameter داریم، $\pi_{observation}$ و learning_rate.

با آزمون و خطا، learning_rate های ثابت با مقادیر ۰.۰۵ , ۰.۱ و ۰.۲ امتحان شدند اما در نهایت یک learning_rate هموگرافیک انتخاب شد که متناسب با تعداد trial ها تغییر می کند. $\pi_{observation}$ ثابت با مقادیر 0.1، 0.05 و 0.2 امتحان شدند اما در نهایت یک learning_rate هموگرافیک انتخاب شد که متناسب با تعداد trial ها تغییر می کند.

نتایج:

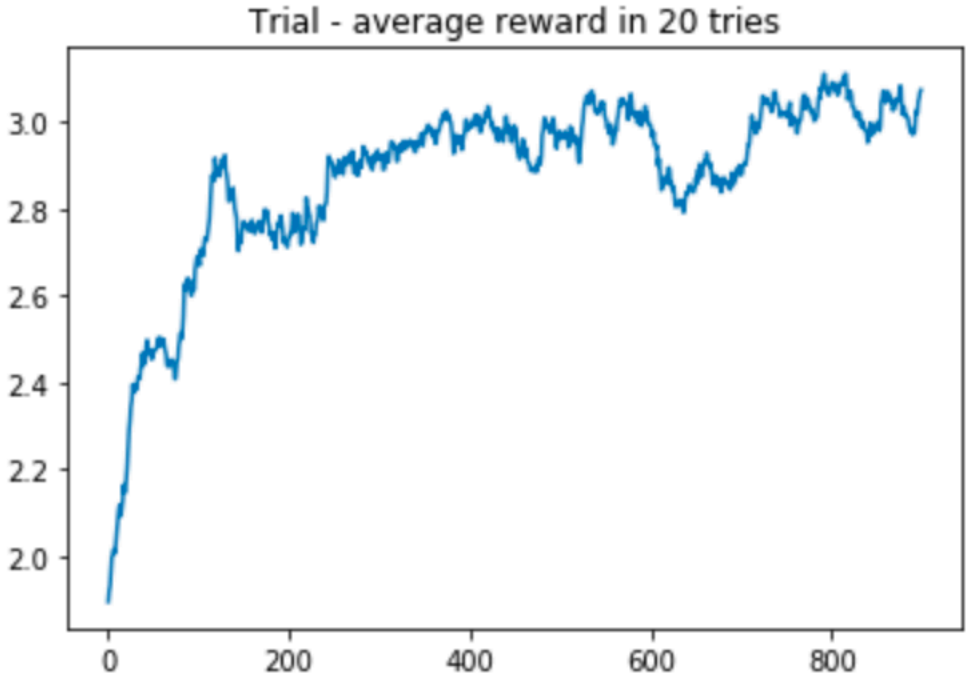
چون در این مسئله عمل بهینه و مقدار میانگین پاداش آن را می دانیم از آنها در محاسبه ی درصد استفاده از عمل بهینه و regret استفاده می کنیم.در مسئله ای که این مقادیر نامعلوم هستند می‌توان این معیارها را بر حسب عملی که در نهایت الگوریتم به عنوان عمل بهینه پیش نهاد می‌کند محاسبه کنیم.





همانطور که قابل مشاهده است پس از حدوود ۵۰۰ مشاهده ، الگوریتم به optimal action رسیده است و درصد استفاده از آن به ۱۰۰ نزدیک شده است .نمودار regret پس از رسیدن به optimal policy همچنان در حال افزایش است . دلیل ایتن موضوع این است که با وجود دست یافتن به optimal action ، پاداش ها خود از یک توضیع نرمال می آیند و نمی توان انتظار داشت که این نمودار کاملا ثابت شود.

همچنین اختلاف این الگوریتم و الگوریتم کاملا رندوم در نمودار های فوق قابل مشاهده است. اگر همین فرآیند را ۲۰ بار انجام دهیم و میانگیری کنیم خواهیم داشت:




Trial - average reward in 20 tries
3.0
2.8
2.6
2.4
2.2
2.0
0
200
400
600
800

سوال دوم

این مسئله در حقیقت یک 10-armed-bandit است. ما می‌توانیم بین یک الی ده دقیقه برای اتوبوس صبر کنیم و درنهایت تاکسی بگیریم تا به کلاس برسیم. پس ۱۰ تصمیم داریم:

۱. یک دقیقه صبر کنیم و اگر اتوبوس نیامد تاکسی بگیریم.

۲. دو دقیقه صبر کنیم و اگر اتوبوس نیامد تاکسی بگیریم.

...

۱۰. ده دقیقه صبر کنیم و اگر اتوبوس نیامد تاکسی بگیریم.

فرض می‌کنیم که حتما میخواهیم به کلاس برسیم بنابر این پس از ۱۰ دقیقه با تاکسی میرویم.

حال باید یک utility function تعریف کنیم تا الگوریتم هایمان توسط آن optimal policy زا پیدا کنند. سه پارامتر در utility function تاثیر دارد:

۱. مدت زمانی که زود رسیدیم (هرچه بیشتر باشد بهتر است.)

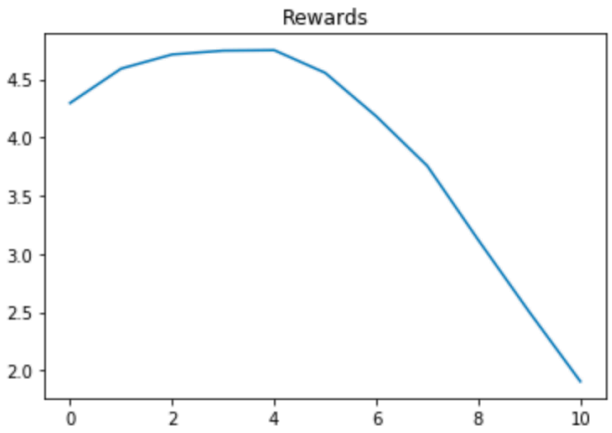
۲.مدت زمانی که منتظر اتوبوس مانده ایم (هرچه کمتر صبر کرده باشیم بهتر است)

۳. استفاده ازتاکسی (اگر از تاکسی استفاده کنیم utility را کم می‌کند.)

تابع utility به صورت زیر تعریف شده است:

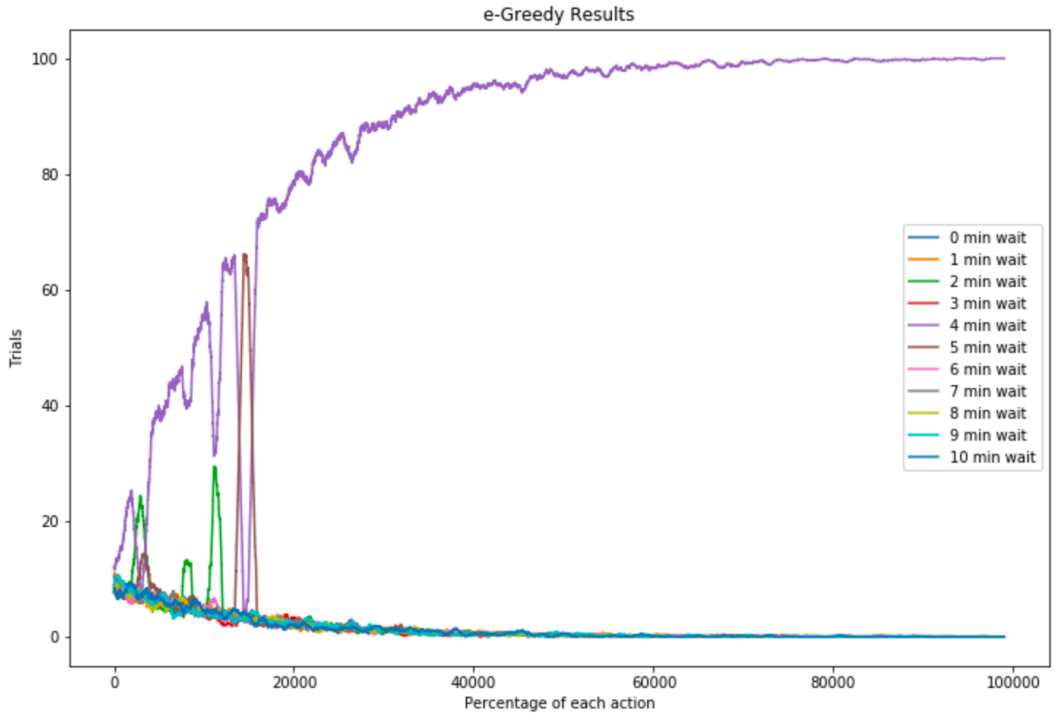
```
def utility(action):
    reward = None
    wait = int(np.random.normal(6, 4, 1))
    if wait < 0: wait = 0
    if wait <= action:
        reward = (10 - wait) * 16 - wait * 12
    else:
        reward = ((10 - action) * 16 - action * 12 ) * 0.2
    return reward / 10
```

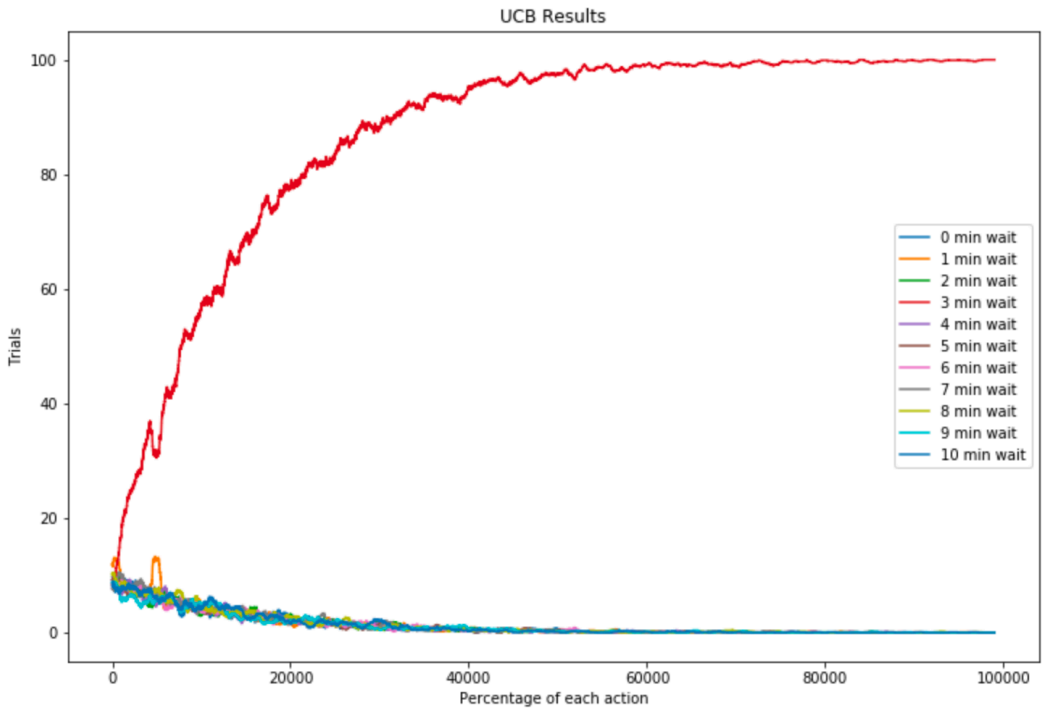
برای اینکه تصور بهتری از تابع utility داشته باشیم آن را برای تعداد زیادی sample برای هر action حساب کرده و میانگین گیری می‌کنیم تا نمودار زیر حاصل شود.



طبق این نمدار، احتمالا action های ۳ یا ۴ (۳ یا ۴ دقیقه صبر کردن برای اتوبوس) بهینه هستند.

الگوریتم های e-greedy و UCB در این قسمت پیاده‌سازی شده اند و نتایج آن‌ها (نمودار درصد ایتفاده از action ها بر حسب trial) در ادامه آمده اند:



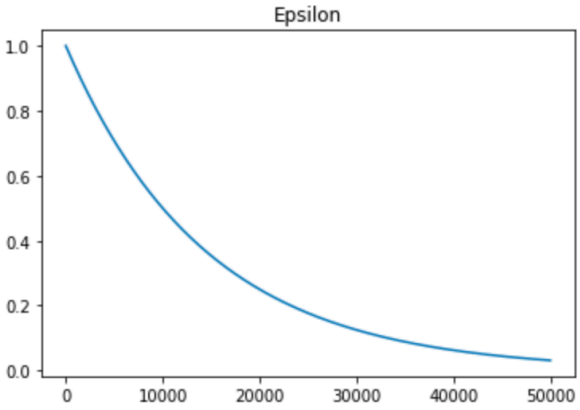


این دو الگوریتم هردو به صورت کورکورانه عمل می‌کنند و در صورتی که hyperparameter های متناسب نباشند، ممکن است حتی به جواب sub-optimal برسند. طبق مشاهدات فوق الگوریتم e-greedy در ۴۰۰۰۰ و الگوریتم UCB در حدود ۳۰۰۰۰ به یک action بهینه می‌رسد. در الگوریتم e-greedy اگر learning rate زیاد باشد با e با نرخ زیادی کم شود ممکن است خیلی زود به جواب های sub-optimal برسیم و این اکگوریتم ذاتا توانایی کنترل این hyper parameter ها را ندارد در صورتی که الگوریتم UCB به طور خودکار action هایی که کمتر انتخاب شده اند یا از حدود q_* آن‌ها اطمینان کمتری دارد را بیشتر انتخاب می‌کند تا تا به optimal policy برسد.

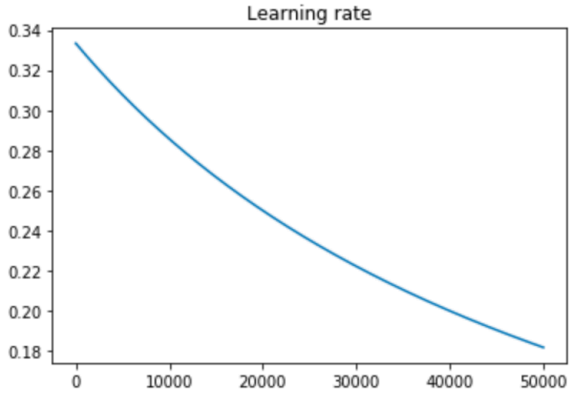
در قسمت ت برای الگوریتم e-greedy از طریق آزمون و خطا چند مقدار مختلف برای learning rate امتحان شد (۰.۰۱, ۰.۰۵, ۰.۱, ۰.۲, ۰.۵) که در طول اجرا ثابت می‌ماندند که در نهایت مقادیر حدود ۰.۰۵ زودتر به جواب می‌رسیدند و درصد جواب های sub-optimal کمتر می‌شد.

در نهایت learning rate یک مقدار متغیر با زمان تعریف شد (۱ به روی تعداد تکرار عمل iام) که نتنها با تعداد trial ها تغییر میکند بلکه برای هر action نیز متفاوت است و عملی که کمتر تکرار شده learning rate بیشتری دارد.

در e-greedy نیز e طبق نمودار زیر با افزایش trial تغییر می‌کند:



در الگوریتم UCB برای learning rate از تابع هموگرافیک زیر استفاده شده است.

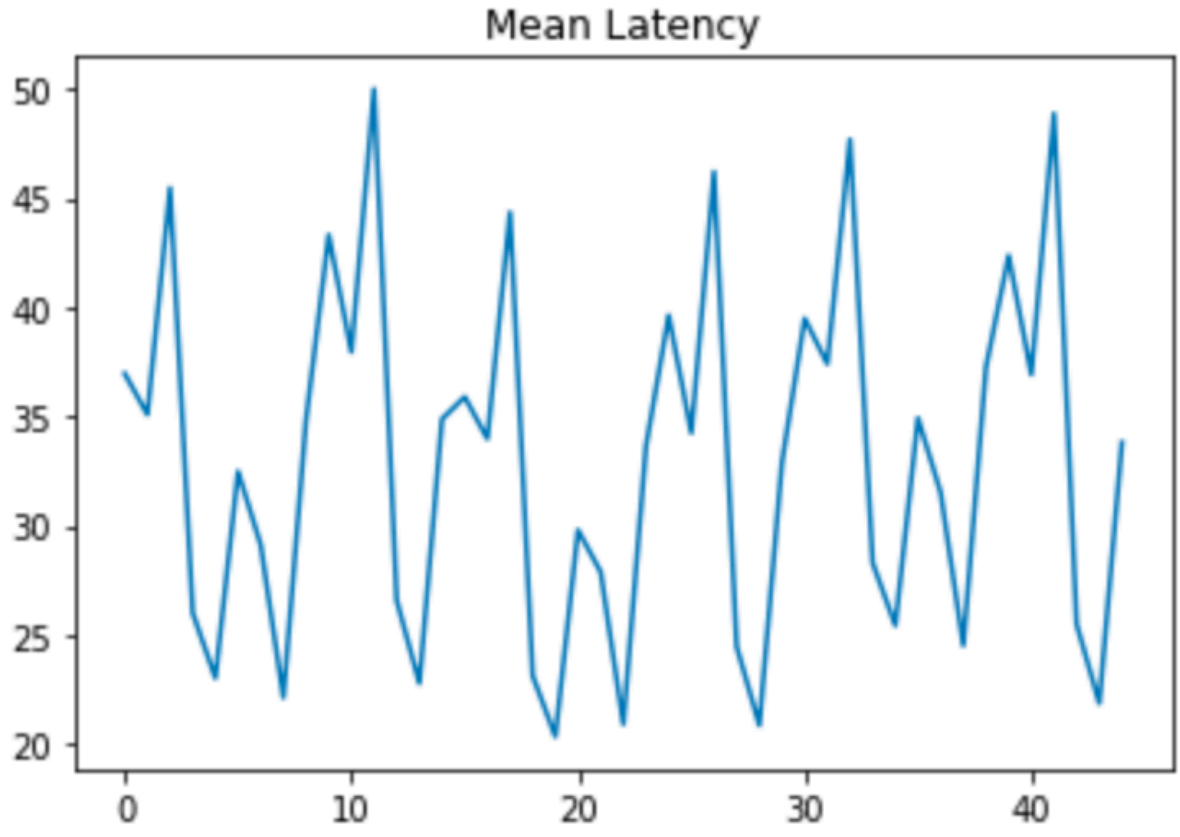


سوال سوم

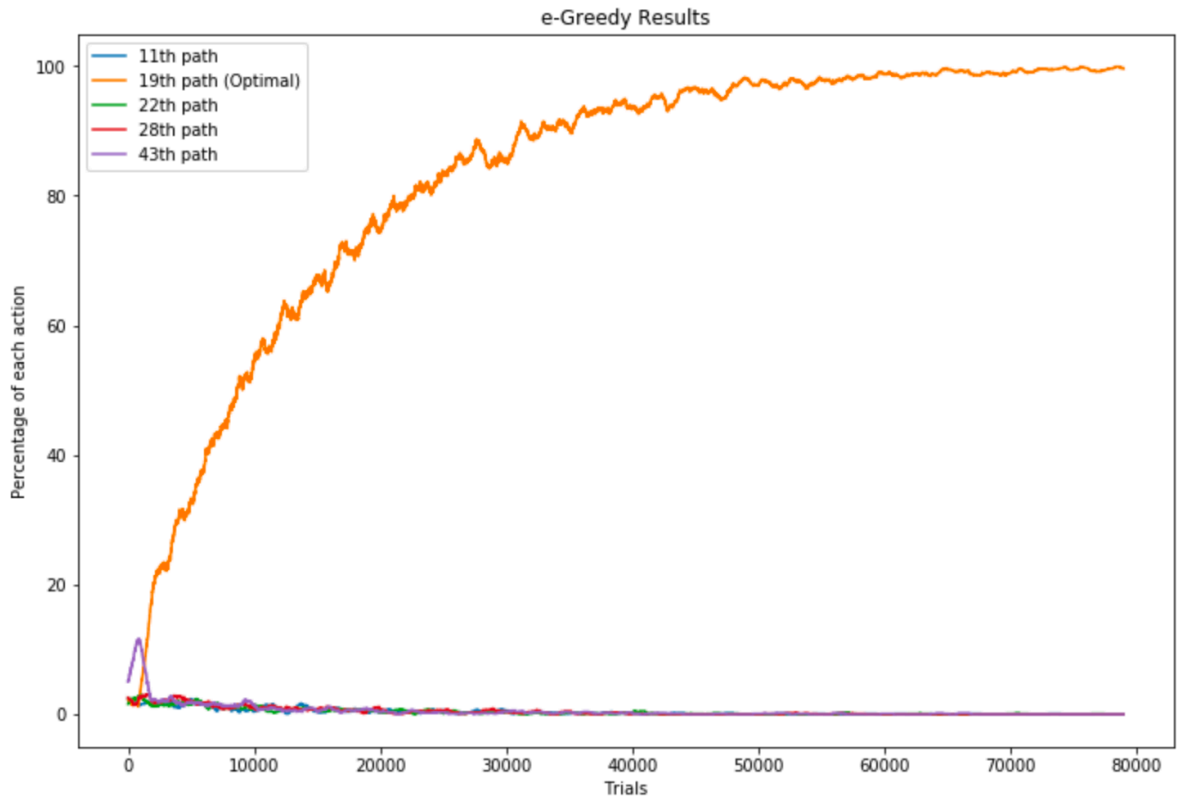
همانند سوال قبل برای حل این سوال به عنوان یک مسئله تصمیم‌گیری نیاز داریم reward و action ها را تعیین کنیم. تعداد action ها برابر است با $1 \times 3 \times 5 \times 3 \times 1 = 45$ (حاصلضرب تعداد node های هر لایه) که برابر با نمام مسیر های ممکن از source به sink است.

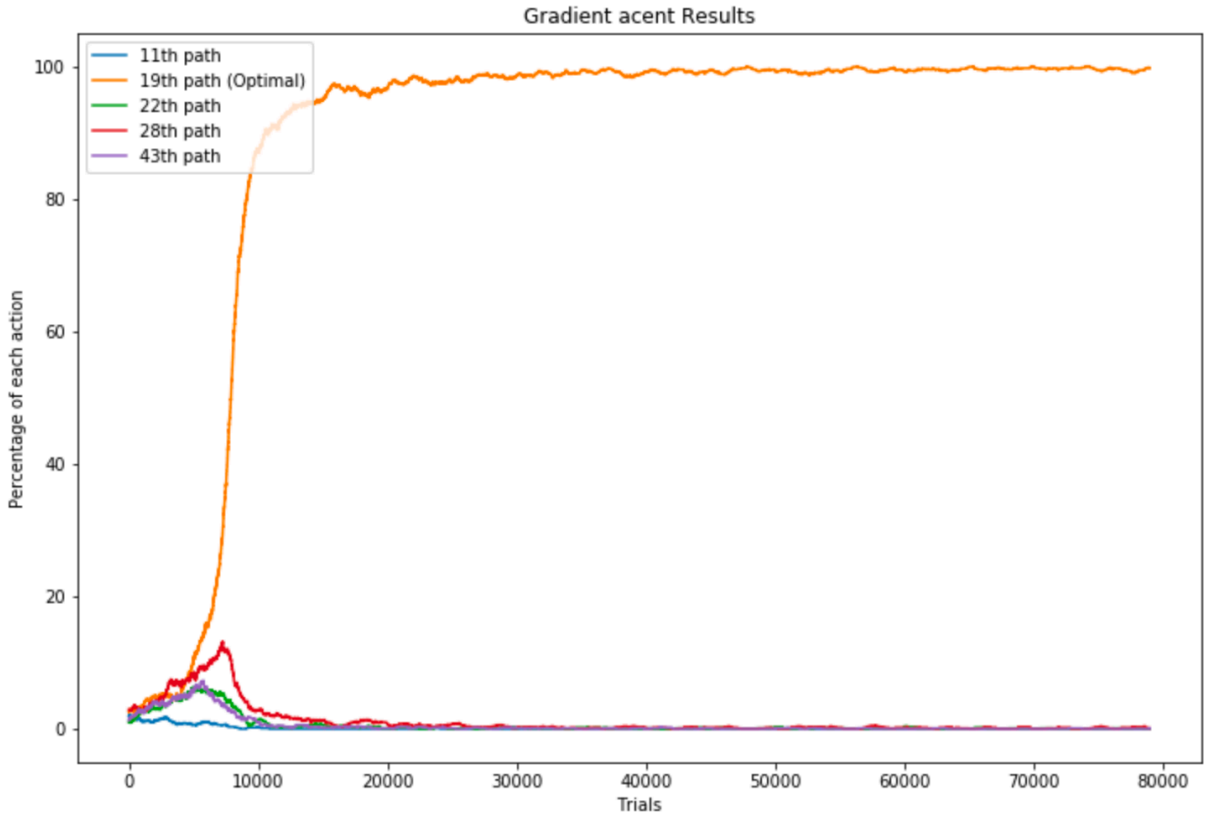
تابع پاداش متناسب است با (constant - latency) که مقدار latency برای هر مسیر برابر است با مجموع مجموع تاخیر ها روی node ها یا یال ها.

همانند مسئله قبل برای تصور بهتر latency هر مسیر، از هر کدام تعداد زیادی sample گرفته شده و میانگسن‌گیری شده و در نمودار زیر قابل مشاهده است:



همانند مسئله دوم، الگوریتم e-greedy با e و learning rate متغیر با زمان استفاده شده و نتایج در ادامه قابل مشاهده اند. الگوریتم gradient ascent نیز با همین learning rate پیاده‌سازی شده.


e-Greedy Results
11th path
19th path (Optimal)
22th path
28th path
43th path
Percentage of each action
Trials
0
20
40
60
80
100
0
10000
20000
30000
40000
50000
60000
70000
80000


Gradient acent Results
11th path
19th path (Optimal)
22th path
28th path
43th path
Percentage of each action
Trials
0
20
40
60
80
100
0
10000
20000
30000
40000
50000
60000
70000
80000

الـگوریتم gradient ascent در کمتر از ۱۰۰۰۰ و الـگوریتم e-greedy در حـدود ۲۵۰۰۰ مـشاهـده عـمل بهینه را پیدا می‌کنند.

بین تمام الگوریتم‌ها gradient ascent از همه مناسب‌تر است و در زمان کمتر می‌تواند به optimal policy برسد. سایر الگوریتم ها اغلب به صورت کورکورانه عمل می‌کنند در حالی که روش gradient ascent با مشتق گیری در فضای preference ها، مسئله تصمیم گیری را به یک مسئله optimization ریاضی تبدیل می‌کند و گرادیان ها نرخ انتخاب action ها و تغییر preference ها را تعیین می‌کنند. Thompson sampling و UCB با تقریب زدن آماره ها برای action ها سعی می‌کنند action ها را آگاهانه تر انتخاب کنند (به طوری که یادگیری سریع‌تر انجام شود و action هایی که کمتر انتخاب شدند یا از اطمینان کمتری به آماره های آن‌ها داریم بیشتر انتخاب شوند) در صورتی که e-greedy اینگونه عمل نمی‌کند و برای این موضوع راه‌حلی ندارد.