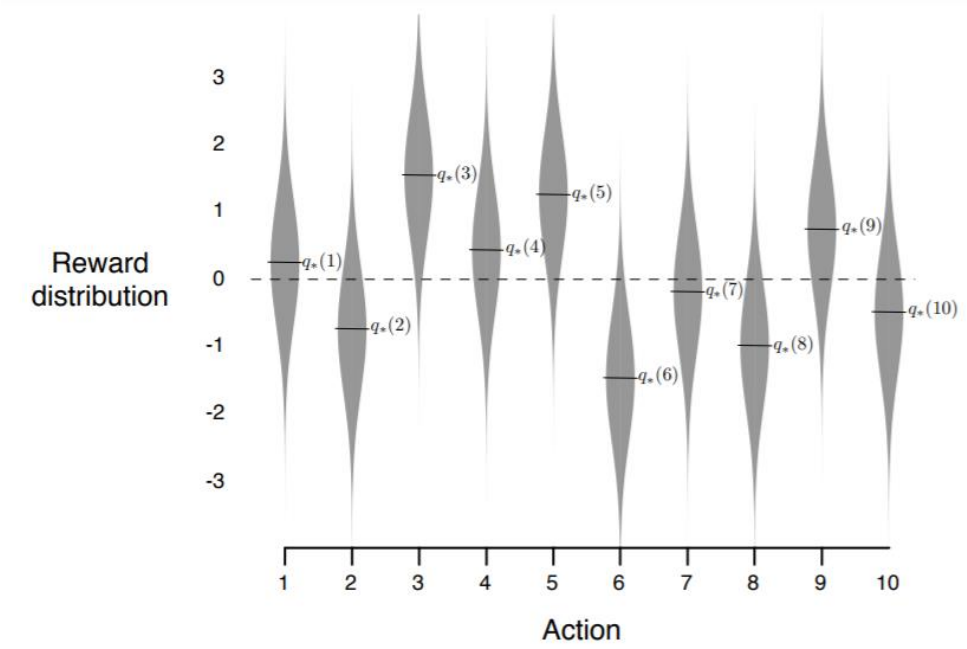
گزارش تمرین شماره ۲

نام و نام خانوادگی	علی عدالت
شماره دانشجویی	۸۱۰۱۹۹۳۴۸

سوال 1 – multi armed bandit شکل ۲.۱ کتاب

در این جا ما یک مسئله multi armed bandit با ۱۰ دسته داریم که توزیع پاداش هر دسته در شکل کتاب آمد است. تمام این توزیع‌ها واریانسی برابر یک دارند و میانگین آن‌ها به صورت تقریبی به شکل زیر است.




```
means = [0.2, -0.8, 1.4, 0.4, 1, -1.5, -0.2, -1, 0.9, -0.4]
```

برای استفاده از الگوریتم thompson sampling در ابتدا برای توزیع‌ها یک واریانس و میانگین اولیه در نظر می‌گیریم. چون اطلاعی از توزیع‌ها در agent وجود ندارد، میانگین تمام توزیع‌ها را صفر قرار دادم و precision را که یک بر روی واریانس است را ۰.۰۰۰۱ قرار دادم تا عدم اطمینان از این توزیع‌ها را به خوبی نشان دهد. در این الگوریتم باید توزیع‌ها را از طریق دریافت پاداش‌ها به روز کنیم. برای این کار از قاعده‌ی بیز استفاده کردم که نحوه‌ی به روز کردن برای یک اکشن در زیر آمده است. در این جا این فرض را داریم که توزیع پاداش‌ها نرمال است.

$$\tau_0 \leftarrow \tau_0 + n\tau$$

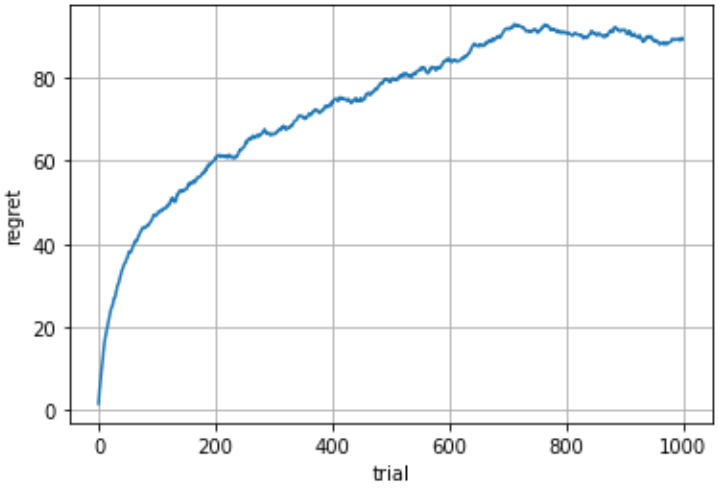
$$\mu_0 \leftarrow \frac{\tau_0\mu_0 + \tau\sum_{i=1}^{n} x_i}{\tau_0 + n\tau}$$

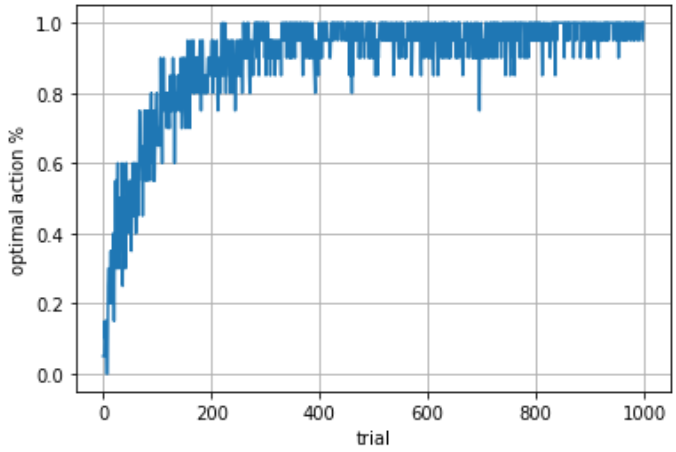
که τ برابر precision پاداش به شرط توزیع اولیه پاداش اکشن است. همچنین μ_0 , τ_0 به ترتیب میانگین و precision توزیع اولیه تخمینی هستند. در این جا با توجه به شکل کتاب ما می‌دانیم که توزیع پاداش‌های دریافتی برابر یک است و به همین دلیل $\tau = 1$ است. با این موضوع قاعده‌ی به روز کردن به شکل زیر است.

$$\tau_0 \coloneqq \tau_0 + 1$$

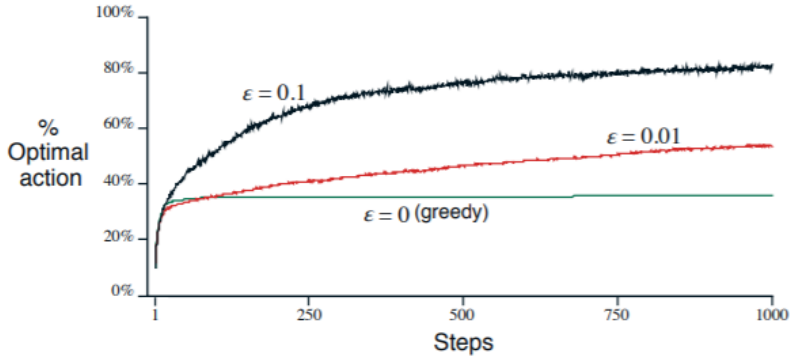
$$\mu_0 \coloneqq \mu_0 + (\frac{1}{1+\tau_0} \times (r - \mu_0))$$

برای انتخاب اکشن نیز از توزیع هر یک از اکشن‌ها یک نمونه می‌گیریم و اکشن با نمونه بزرگتر را انتخاب می‌کنیم. با انجام تعداد مناسبی اکشن در exploration توزیع‌های اصلی تقریبا بدست می‌آیند و از آن به بعد ما اکشن با متوسط بیشینه را انتخاب می‌کنیم و exploitation انجام می‌دهیم. نمودار regret و در صد استفاده از اکشن بهینه برای ۲۰ اجرا که هر یک شامل ۱۰۰۰ آزمایش است در زیر آمده است.

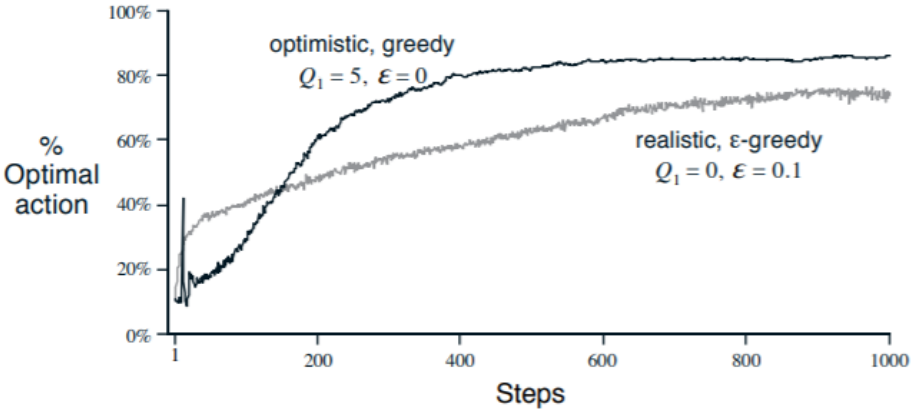




میزان regret تجمعی متوسط در افق ۱۰۰۰ برابر ۸۹.۲۰۲ است. در زیر نمودار کتاب با الگوریتم epsilon greedy آمده است.

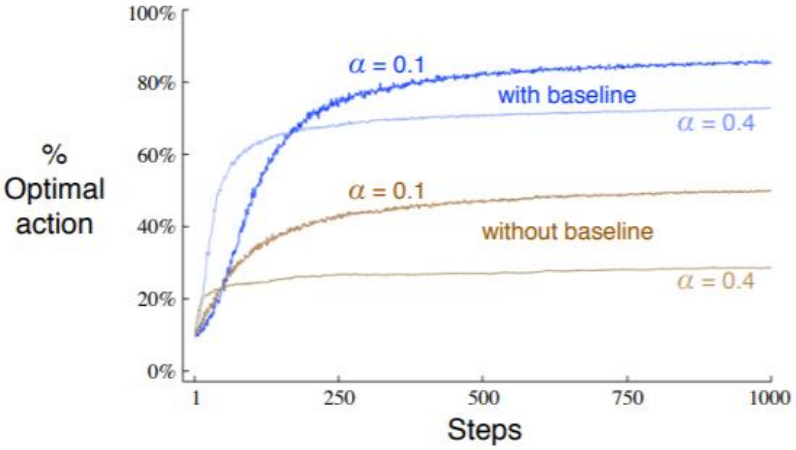


همانطور که دیده می‌شود، الگوریتم در افق ۱۰۰۰ نتوانسته به ۱۰۰ درصد انتخاب اکشن بهینه برسد این در حالی است که thompson sampling به این درصد رسیده است. در نمودار بالا تقریبا بعد از ۵۰۰ گام نمودار به ثبات نزدیک شده است این درحالی است که در نمودار مربوط به thompson sampling الگوریتم در ۴۰۰ امین گام بر روی مقدار ۱۰۰ درصد ثابت شده است. این موضوع نشان می‌دهد که thompson sampling سریعتر توانسته اکشن بهینه را پیدا کند و از exploration به exploitation تغییر وضعیت دهد. این مورد باعث می‌شود که مجموع پاداش دریافتی ما نیز بیشتر شود. این موضوع نشان دهنده‌ی عملکرد بهتر الگوریتم Thompson sampling است. در زیر نمودار دیگری را برای الگوریتم epsilon greedy با و بدون استفاده از مقداردهی اولیه optimistic مشاهده می‌کنید.



در این جا نیز مشاهده می‌کنید که الگوریتم epsilon greedy با و بدون مقدار دهی خوشبینانه، در حدود ۵۰۰ گام به بالا به مقدار ثابت درصد استفاده از اکشن بهینه می‌رسد که از ۴۰۰ گام الگوریتم Thompson بیشتر است. همچنین الگوریتم egreedy باز هم به صد درصد استفاده از اکشن بهینه نرسیده است. در این نمودار باز مشاهده می‌شود که thompson sampling زودتر توانسته اکشن بهینه را بیابد و به

exploitation بپردازد. این مورد باعث می‌شود جمع پاداش دریافتی ما بیشتر هم بشود. این موضوع نشان دهنده‌ی عملکرد بهتر الگوریتم Thompson sampling است. نمودار دیگری نیز در کتاب وجود دارد که مربوط به gradient method است که در زیر آمده است. نکاتی که در نمودارهای قبل دیدیم در این نمودار نیز قابل مشاهده است. Thompson توانسته سریع تر اکشن بهینه را پیدا کند که این موضوع نشان می‌دهد عملکرد آن از gradient method بهتر بوده است.



سوال ۲ – تعیین زمان صبر کردن در ایستگاه اتوبوس

الف) در این مسئله برای این که ما به موقع به کلاس ساعت ۷:۳۰ برسیم، حداکثر می‌توانیم ۱۵ دقیقه در ایستگاه صبر کنیم. در غیر این صورت با توجه به مسافت ۲۰ دقیقه‌ای تا مقصد در هر حالت تاخیر خواهیم داشت. در این جا ما رزولوشن زمان صبر کردن را یک دقیقه در نظر می‌گیریم. دلیل این موضوع این است که صبر کردن زیر یک دقیقه معنی ندارد. همچنین صبر زیر یک دقیقه مثل ۳۰ ثانیه با توجه به مسافت تاکسی‌ها از اتوبوس معنی نخواهد داشت. همچنین ما حداقل یک دقیقه صبر می‌کنیم چرا که هزینه تاکسی بیشتر از اتوبوس است. ما با یک مسئله n-armed bandit مواجه هستیم که فقط یک state داریم. در آن state زمانی را برای اتوبوس صبر می‌کنیم. در این جا ما توزیع احتمال زمان رسیدن اتوبوس را نمی‌دانیم و می‌خواهیم با چندین بار قرار گرفتن در این state ، زمان صبر برای اتوبوس را تعیین کنیم. با توجه به رزولوشن یک دقیقه، اکشن‌ها را به صورت زیر تعریف می‌کنیم. در کل ما ۱۵ اکشن داریم که مجموعه‌ی آن‌ها را A می‌نامیم.

$$a_i = \ wait\ i\ minute\ and\ then\ take\ a\ taxi$$

بعد از انجام اکشن، ما reward را دریافت می‌کنیم. با توجه به زمان رسیدن اتوبوس که زمان لازم برای صبر کردن ما بوده‌است، reward اکشن تعیین می‌شود. تابع reward برای هر اکشن به صورت زیر است.

$$r(a_i) = \begin{cases} -5000, & i < s \\ 0, & i \geq s \end{cases}$$

در این جا s زمان رسیدن اتوبوس است و i زمانی است که ما برای اتوبوس صبر کرده‌ایم. s از یک توزیع نرمال با میانگین ۸ و انحراف از معیار ۳ به صورت رندم در ازای انجام هر اکشن انتخاب می‌شود. اگر ما کمتر از زمان لازم صبر کنیم، هزینه‌ی ۵۰۰۰ تومان بیشتر برای تاکسی پرداخته‌ایم. اگر به اندازه‌ی کافی صبر کنیم، هزینه‌ای بر ما تحمیل نمی‌شود. در این جا فرض کرده‌ایم که صبر در ایستگاه در مدت زمان مورد قبول ما، هزینه‌ای ندارد.

ما انسان‌ها به طور معمول از هزینه‌ها دوری می‌کنیم و آن‌ها را بزرگتر از واقیت می‌دانیم. همچنین از میزان هزینه‌ای به بعد، تغییرات هزینه به اندازه‌ی کمتری در ذهن ها حساسیت ایجاد می‌کند. با توجه به این نکته تابع utility به صورت زیر خواهد بود. ساختار این تابع ویژگی‌های نگاه انسان به هزینه را برآورده می‌کند. توان باید کمتر از یک باشد تا ویژگی‌های گفته شده را داشته باشیم. بر اساس پرسش از تعدادی دانشجو و تعیین ارزش واقعی پاداش‌ها از نظر آن‌ها، این ضریب و توان بدست آمده است.

$$u(a_i) = \ -800 \times |r(a_i)|^{0.3}$$

در این مسئله فرض کردیم که زودتر رسیدن به دانشگاه برای ما هزینه‌ای ندارد و وقت ما هدر نمی‌رود.

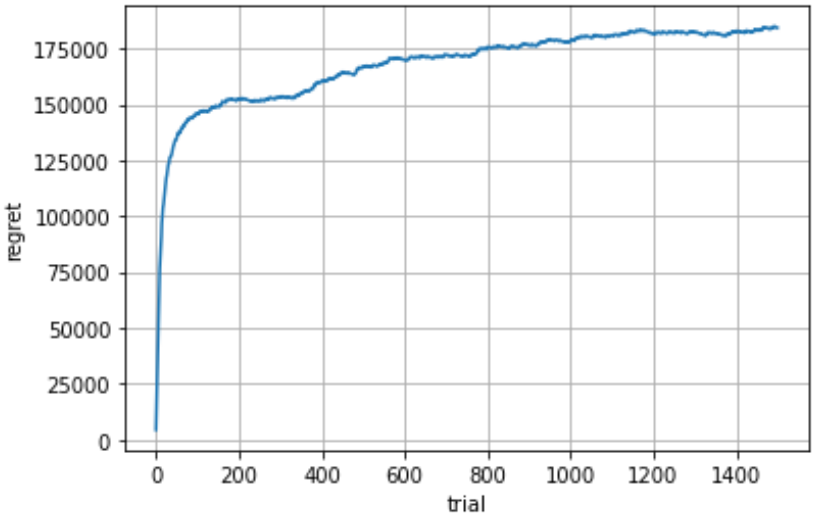
ب) در این بخش به پیاده‌سازی و آموزش می‌پردازیم تا عامل، زمان بهینه صبرکردن را یاد بگیرد. برای این یادگیری از سیاست epsilon-greedy استفاده می‌کنیم که epsilon به صورت $\frac{1}{n}$ تغییر می‌کند که n شماره trial است. در طول یادگیری متوسط پاداش هر اکشن را به صورت point estimation تخمین می‌زنیم و بر اساس آن اکشن بهینه a^* را انتخاب می‌کنیم. اکشن بهینه، اکشنی است که متوسط پاداش بیشتری داشته باشد. در این صورت احتمال انتخاب هر اکشن در trial به صورت زیر است.

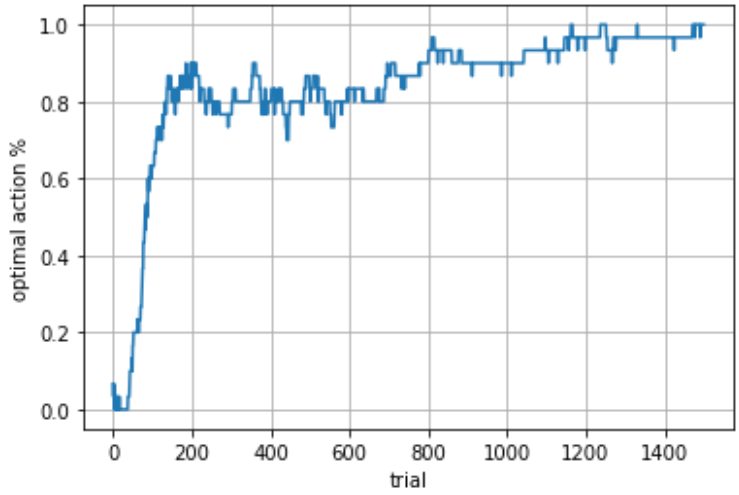
$$p(a_i) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{\|A\|}, & a_i = a^* \\ \frac{\varepsilon}{\|A\|}, & a_i \neq a^* \end{cases}$$

برای تخمین متوسط پاداش هر اکشن، از میانگین پاداش‌های گرفته شده برای آن استفاده می‌کنیم. در زیر این موضوع بعد از انجام n بار اکشن a_i آمده است.

$$Q(a_i) = \frac{\sum_{j=1}^{n} u_j(a_i)}{n}$$

برای آموزش agent تعداد ۱۵۰۰ آزمایش در هر بار اجرا انجام دادیم. برای رسیدن به نتیجه‌ی معتبر با توجه به stochastic بودن، ۳۰ بار اجرا انجام دادیم. برای این که الگوریتم در دام اکشن غیر بهینه گیر نکند و آموزش به خوبی انجام شود، از optimistic initialization استفاده می‌کنیم. مقدار $Q(a_i)$ های اولیه را برابر صفر قرار می‌دهیم. نمودار درصد استفاده از اکشن بهینه و regret در زیر آمده است.



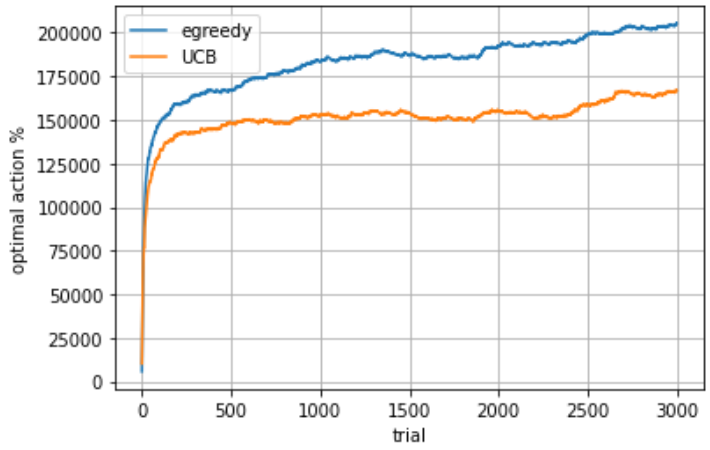


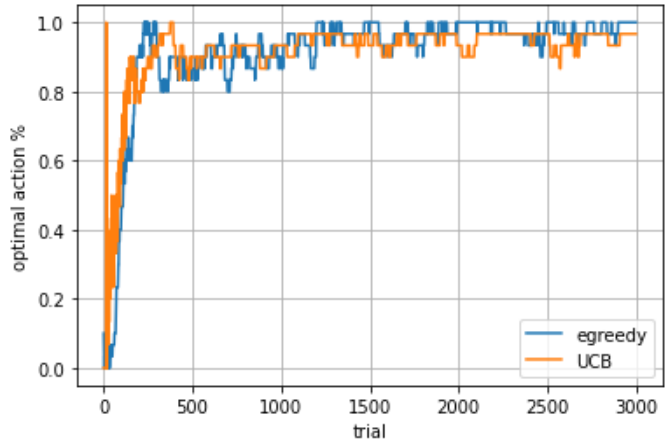
بعد از آموزش agent این نتیجه بدست آمد که اکشن بهینه a_{14} یا ۱۵ دقیقه صبر کردن است.

پ) در این‌جا agent با سیاست UCB را توسعه می‌دهیم. برای انتخاب اکشن بهینه، به صورت زیر عمل می‌کنیم.

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right]$$

برای محاسبه و به روز رسانی $Q(a_i)$ مانند الگوریتم epsilon greedy عمل می‌کنیم و میانگین پاداش‌های گرفته شده برای آن اکشن را محاسبه می‌کنیم. برای انتخاب اکشن بهینه مانند بالا عمل می‌کنیم. c مقدار اطمینان است که برای کنترل exploration از آن استفاده می‌شود و $N_t(a)$ بیانگر تعداد بار استفاده از اکشن a است. عبارت رادیکالی در این جا بیانگر میزان exploration است. UCB به جای تخمین میانگین، توزیع پاداش اکشن‌ها در نظر می‌گیرد. این موضوع باعث می‌شود که در exploration عملکرد بهتری از epsilon greedy داشته باشد. در زمان exploration همانطور که در بالا در بخش رادیکالی دیده می‌شود، هر چه اکشنی کمتر انجام شده باشد برای انتخاب احتمال بالاتری خواهد داشت. از طرفی با دیدن اکشن‌های بیشتر واریانس توزیع‌ها کمتر می‌شود. با گذشت زمان و کم شدن واریانش توزیع‌ها، اکشن‌های با متوسط کمتر، احتمال انتخابشان کمتر می‌شود. این موضوع باعث می‌شود عملکرد UCB بهتر باشد. در زیر نمودار درصد استفاده از اکشن بهینه و regret برای دو الگوریتم UCB و epsilon greedy در کنار هم آمده است.

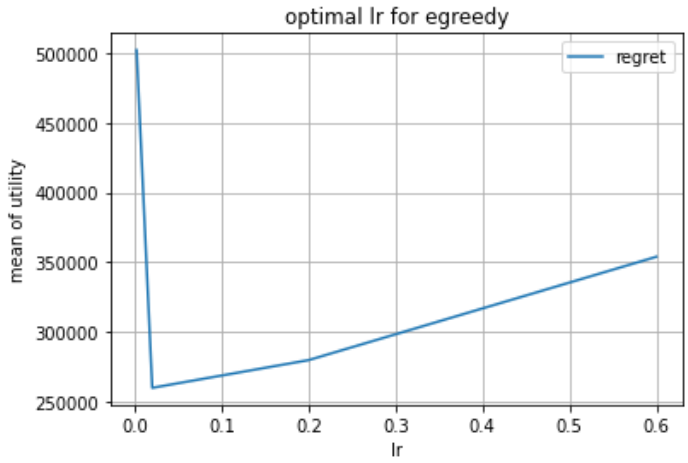




در هر دو الگوریتم، ۱۵ دقیقه صبر کردن به عنوان اکشن بهینه انتخاب شده است. همان طور که در نمودار regret هر دو الگوریتم دیده می‌شود، میزان regret و اندازه‌ی متوسط utility دربافتی در الگوریتم UCB کمتر از الگوریتم epsilon greedy است. در این مسئله، utility منفی است و هر چه اندازه‌ی آن کمتر باشد، الگوریتم بهینه تر است. به همین دلیل کمتر بودن regret که به اندازه utility کار دارد، بهتر است. هم چنین دیده‌می‌شود که نمودار UCB به شکل log است و زودتر به مقدار نهایی regret می‌رسد. با بررسی نمودار درصد استفاده از اکشن بهینه دیده می‌شود که UCB با شیبی کمی بیشتر از دیگر الگوریتم به حالت ثبات در انتخاب اکشن بهینه می‌رسد. همچنین قبل از رسیدن به ثبات در لحظه‌ای به انتخاب ۱۰۰ درصد اکشن بهینه رسیده است اما همچنان به انتخاب دیگر اکشن‌ها ادامه داده است تا exploration تمام شود. الگوریتم epsilon greedy از نظر پیاده سازی و منطق ریاضی بسیار ساده است و با همین شرایط، عملکرد مناسبی نیز دارد. یک عیب در این الگوریتم، عملکرد غیر بهینه در زمان exploration است. الگوریتم در این زمان بصورت راندم عمل می‌کند. الگوریتم UCB پیچیده‌تر از epsilon greedy است و بهینه تر به exploration می‌پردازد. در الگوریتم epsilon greedy احتمال انتخاب اکشن ها با متوسط پاداش تخمینی کمتر با هم برابر است. این موضوع باعث می‌شود اکشنی که پاداش خیلی پایینی هم دارد

را در exploration به تعداد زیادی انتخاب کنیم. در UCB با در نظر گرفتن توزیع پاداش اکشن‌ها، اکشن بهینه انتخاب می‌شود. در این شرایط اگر اکشنی میانگین خیلی پایینی داشته باشد، دیگر در exploration انتخاب نمی‌شود. این موضوع باعث کمتر شدن regret و اندازه‌ی متوسط utility دریافتی می‌شود که در بالا می‌بینیم. در UCB با زیاد شدن انجام اکشن و رسیدن به داده‌ی کافی، واریانس توزیع پاداش اکشن‌ها کمتر می‌شود و با جلو رفتن زمان از exploration به exploitation می‌رویم و از آن به بعد به صورت greedy عمل می‌کنیم. در epsilon greedy ما میانگین پاداش اکشن‌ها را فقط تخمین می‌زنیم و در حالت epsilon ثابت، در طول زمان نمی‌توانیم از explore به exploitation بپردازیم. در UCB با تعیین سطح اطمینان می‌توان میزان exploration را مشخص کرد و این موضوع دست ما است.

ت) در cross validation ما تعدادی داده را به صورت رندم برای train و تست انتخاب می‌کنیم و برای گزارش عملکرد چندین بار این کار را انجام می‌دهیم. نتایج حاصل از تست این چند بار را متوسط می‌گیریم و اعلام می‌کنیم. در اینجا هر بار انجام اکشن به صورت راندم انجام می‌شود. برای train عامل را با مقدار پارامتر مورد نظر می‌سازیم. سپس بصورت رندم آنقدر اکشن انجام می‌دهیم تا تصمیم بهینه در محیط بدست آید. در این جا train به صورت راندم انجام شده است. سپس تعدادی اکشن دیگر انجام می‌دهیم که مانند عمل بر روی داده‌ی تست تصادفی است. متوسط utility حاصل از اکشن‌های تست، نشانگر عملکرد عامل با مقدار خاص learning rate است. برای این موضوع ۲۰۰۰ اکشن برای train و ۵۰۰ اکشن برای تست انجام دادیم. به ازای یک learning rate خاص ۳۰ بار این کار را با عامل انجام می‌دهیم ومتوسط utility این چند بار اجرا را به عنوان عملکرد عامل به ازای آن learning rate در نظر می‌گیریم. در زیر به ازای learning rate های متفاوت، عملکرد عامل با الگوریتم epsilon greedy را می‌بینیم. در این مسئله به دلیل منفی بودن utility، ما نمودار 1- ضرب در utility را در زیر رسم کرده‌ایم. در این شرایط کمینه بودن utility نشان دهنده‌ی بهترین عملکرد است.




```
lrs = [0.002, 0.02, 0.2, 0.6]
```

همانطور که دیده می‌شود به ازای learning rate برابر ۰.۰۲ مقدار اندازه‌ی متوسط utility کمینه است و عملکرد عامل با الگوریتم epsilon greedy ما با learning rate ثابت بهینه است.

سوال 3 – یافتن مسیر با کمترین تاخیر در شبکه

الف) در این جا فرض می‌کنیم که ساختار شبکه شامل گره‌ها و یال‌ها را می‌دانیم و فقط توزیع تاخیر یال‌ها و گره‌ها برای ما مجهول است. در این ساختار گراف، تمام مسیرهای از ۰ به ۱۲ از یکی از گره‌های ۵ یا ۶ یا ۷ عبور می‌کنند. به همین دلیل مسیر بهینه نیز از یکی از این گره‌ها عبور خواهد کرد. فرض کنیم مسیر بهینه از گره ۵ عبور کند. این مسیر شامل مسیر بهینه از گره ۰ به گره ۵ و مسیر بهینه از گره ۵ به ۱۲ است. در غیر این صورت بهینه بودن مسیر از ۰ به ۱۲ زیر سوال می‌رود. به همین دلیل می‌توان مسئله یافتن مسیر بهینه را با این شرایط به ۶ زیر مسئله شکست که در زیر آمده است.

- یافتن مسیر بهینه از ۰ به ۵
- یافتن مسیر بهینه از ۵ به ۱۲
- یافتن مسیر بهینه از ۰ به ۶
- یافتن مسیر بهینه از ۶ به ۱۲
- یافتن مسیر بهینه از ۰ به ۷
- یافتن مسیر بهینه از ۷ به ۱۲

مسئله اصلی را می‌توان اینگونه مدل کرد که عامل بارها در state یافتن مسیر از ۰ به ۱۲ قرار می‌گیرد و اکشن‌های قابل اجرای آن ارسال بر تمام مسیرهای ممکن از ۰ به ۱۲ است. در این شرایط برای مثال یال (۰و۱) در ۱۲ اکشن و یال (۵و۹) در ۴ اکشن تکرار می‌شوند. این تکرارها باعث می‌شوند که تعداد زیادی اکشن بیهوده انجام دهیم تا توزیع این یال‌های تکراری را تخمین بزنیم. با توجه به این که می‌خواهیم تعداد اکشن‌ها را کمینه کنیم، باید تا حد ممکن از تکرار یال‌ها در اکشن‌های ممکن جلوگیری کنیم. به همین دلیل روش بالا را برای شکستن مسئله پیشنهاد دادیم. هر یک از این زیر مسائل به تنهایی با n-armed bandit حل می‌شوند و ما با چندین مسئله‌ی n-armed bandit مواجه هستیم. در ادامه اکشن‌های ممکن در هر زیر مسئله را بیان می‌کنیم. تابع پاداش و utility در تمام این مسائل یکسان است. پاداش هر یک از اکشن‌ها برابر تابعی از تاخیری است که بسته برای رسیدن به مقصد دارد. تابع پاداش در زیر آمده است.

$$r(a_i) = -1 \times latency(a_i)$$

در این گونه موارد، عامل به صورت کاملا منطقی است و با توجه به این نکته تابع utility با reward برابر است. یک مثال برای عامل سوال می‌تواند یک روتر باشد که می‌خواهد مسیر بهینه را پیدا کند که یک عامل کاملا منطقی است.

برای پیدا کردن مسیر بهینه در کل شبکه، مسیر بهینه از ۰ به ۵ را به مسیر بهینه از ۵ به ۱۲ می‌چسبانیم. برای گره‌های ۶ و ۷ نیز همین کار را می‌کنیم. سپس از میان این مسیرهای بدست آمده، مسیر بهینه را انتخاب می‌کنیم. تاخیر مسیر بهینه از ۰ به ۱۲ که از گره ۵ می‌گذرد، از جمع تاخیر مسیر بهینه از ۰ به ۵ و تاخیر مسیر بهینه از ۵ به ۱۲ بدست می‌آید. در زیر اکشن‌های زیر مسئله‌ها آمده است.

یافتن مسیر بهینه از ۰ به ۵

شماره اکشن	مسیر متناظر
۱	۰و۱و۵
۲	۰و۲و۵
۳	۰و۳و۵
۴	۰و۴و۵

یافتن مسیر بهینه از ۵ به ۱۲

شماره اکشن	مسیر متناظر
۱	۵و۸و۱۲
۲	۵و۹و۱۲
۳	۵و۱۰و۱۲
۴	۵و۱۱و۱۲

یافتن مسیر بهینه از ۰ به ۶

شماره اکشن	مسیر متناظر
۱	۰و۱و۶
۲	۰و۲و۶
۳	۰و۳و۶
۴	۰و۴و۶

یافتن مسیر بهینه از ۶ به ۱۲

شماره اکشن	مسیر متناظر
۱	۶و۸و۱۲
۲	۶و۹و۱۲
۳	۶و۱۰و۱۲
۴	۶و۱۱و۱۲

یافتن مسیر بهینه از ۰ به ۷

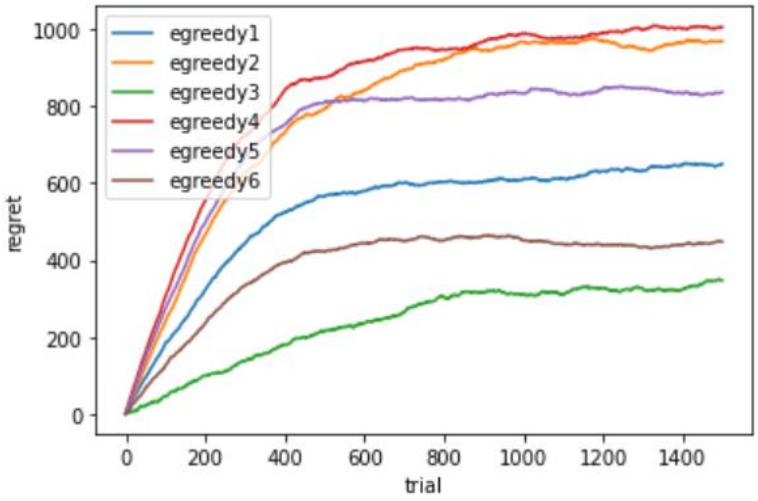
شماره اکشن	مسیر متناظر
۱	۰و۱و۷
۲	۰و۲و۷
۳	۰و۳و۷
۴	۰و۴و۷

یافتن مسیر بهینه از ۷ به ۱۲

شماره اکشن	مسیر متناظر
۱	۷و۸و۱۲
۲	۷و۹و۱۲
۳	۷و۱۰و۱۲
۴	۷و۱۱و۱۲

ب) مانند سوال قبل به حل مسئله با الگوریتم epsilon greedy پرداختیم. در الگوریتم برای این که کمترین تعداد آزمایش را داشته باشیم، epsilon را مانند قبل به صورت $\frac{1}{n}$ تغییر می‌دهیم و learning rate را برابر

۰.۰۲ قرار می‌دهیم. برای حل هر زیر مسئله ۵۰ بار اجرا می‌کنیم تا تاخیر و regret متوسط را با توجه به stochastic بودن مسئله بدست بیاوریم. در هر بار اجرا یک زیر مسئله ۱۵۰۰ اکشن برای یادگیری انجام می‌دهیم. در زیر نمودار regret متوسط برای زیر مسئله‌ها به ازای شرایط بالا آمده است.



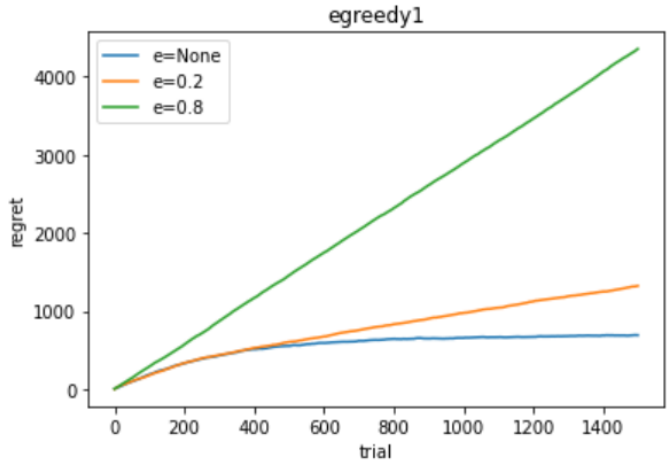
همانطور که دیده می‌شود زیر مسئله‌ها با ۱۵۰۰ اکشن به طور کامل حل می‌شوند و ۱۵۰۰ کمترین تعداد اکشن موردنیاز برای هر زیر مسئله است. با این شرایط آمار مسیر بهینه به شکل زیر است. در این آمار بهترین مسیر، متوسط تاخیر آن و تعداد اکشن‌های انجام شده در هر اجرا ذکر شده است.

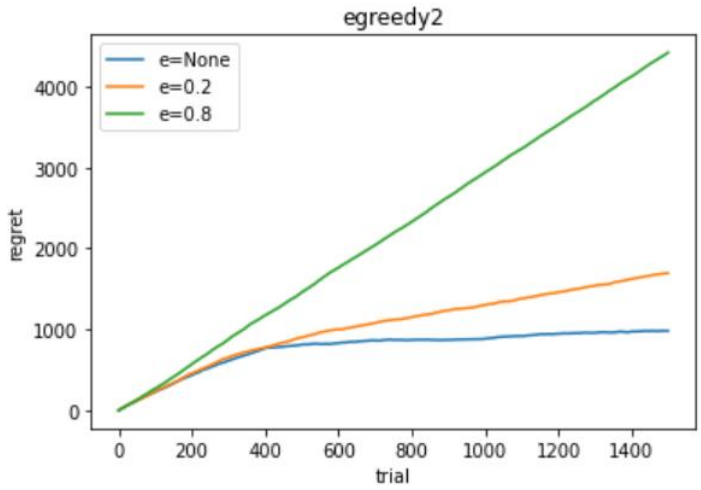
```
best path:  [0, 1, 5, 9, 12]  mean latency: 18.760595506353027  actions:  9000
```

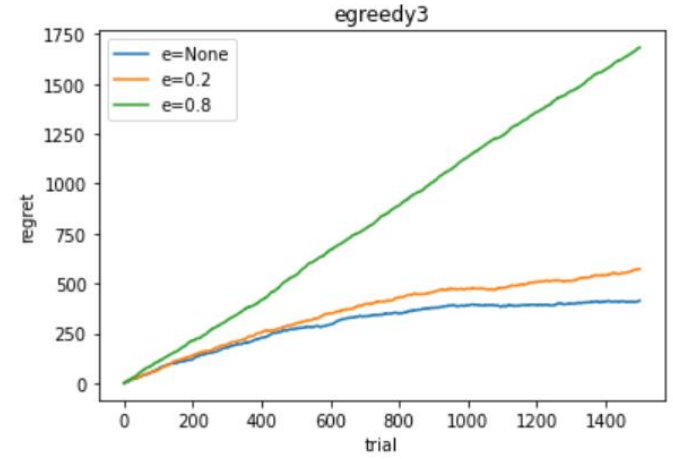
حال هر زیرمسئله را برای حالت epsilon ثابت، برای مقادیر زیر حل می‌کنیم و نمودار regret را برای این مقادیر در کنار هم رسم می‌کنیم. این نمودارها نیز در زیر آمده است. در تمام این نمودارها دیده می‌شود که هر چه epsilon به یک نزدیک می‌شود، عملکرد الگوریتم به سیاست راندم نزدیک تر می‌شود. که این موضوع را می‌توان در نمودارهای regret زیر دید که نمودار regret برای epsilon برابر ۰.۸ به صورت یک خط است مانند نمودار regret سیاست راندم. در حالت epsilon نزدیک به یک، الگوریتم بیشتر به exploration می‌پردازد و تقریبا exploitation نداریم. به همین دلیل regret با زیاد شدن اکشن‌ها زیاد می‌شود. در حالت epsilon نزدیک به صفر، الگوریتم هیچگاه اکشن بهینه را پیدا نمی‌کند که بعد از آن از آن استفاده کند. عامل با احتمال بالاتری از حالت epsilon برابر ۰.۸، اکشن با متوسط بیشتر را انتخاب می‌کند اما هیچ وقت به حالتی نمی‌رسد که احتمال اکشن بهینه نزدیک یک شود. در این حالت عامل با احتمال بیشتری به صورت راندوم عمل می‌کند و regret در زمان افزایش می‌یابد. اما چون احتمال انتخاب

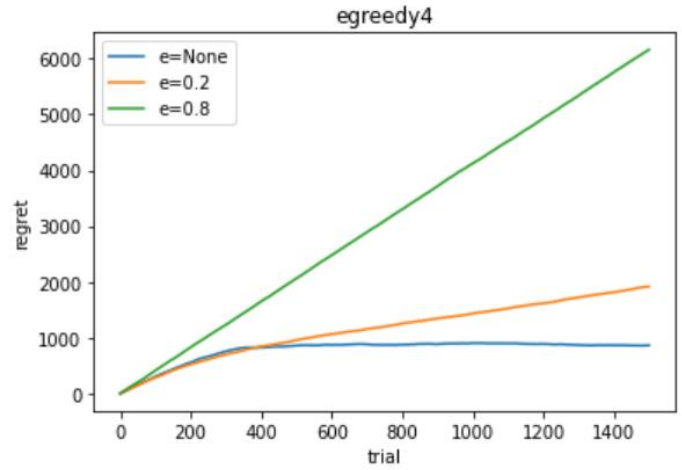
اکشن بهینه بیشتر از قبل است، نمودار در ابتدا به log شبیه است و رشد regret از حالت قبل کمتر است. حالت دیگر در این نمودارها استفاده از epsilon برابر None است که به معنی استفاده از epsilon متغیر با زمان است که در ابتدا برای حل مسئله استفاده کردیم. در این روش در ابتدا به exploration می‌پردازیم و با گذشت زمان و تکمیل اطلاعات به فاز exploitation می‌رسیم و از آن به بعد greedy عمل می‌کنیم. در این روش ما کورکورانه از exploration به exploitation می‌رویم. زیرمسائل مربوط به نمودارها به ترتیب به صورت زیر است.

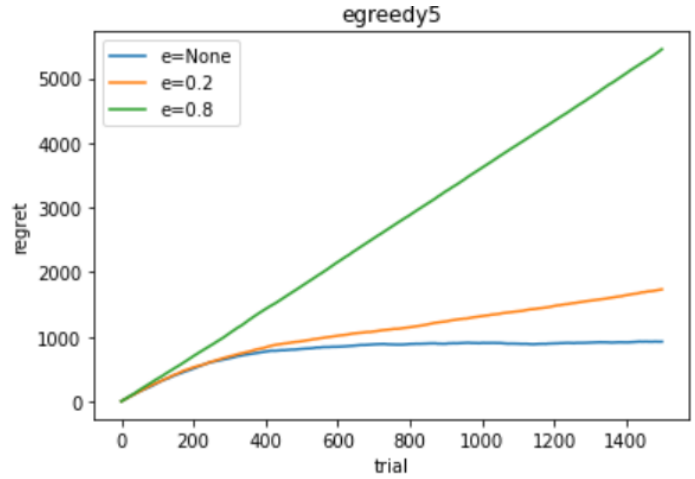
```
[pathes_t5, pathes_t6, pathes_t7, pathes_f5, pathes_f6, pathes_f7]
```

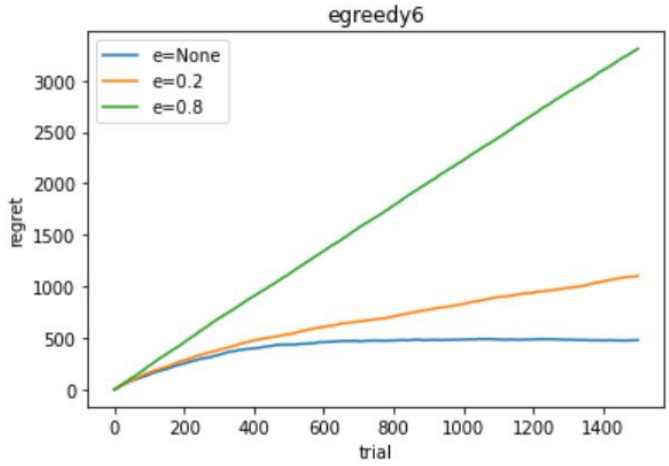





egreedy3
e=None
e=0.2
e=0.8
regret
trial
0
250
500
750
1000
1250
1500
1750
0
200
400
600
800
1000
1200
1400


egreedy4
e=None
e=0.2
e=0.8
regret
trial
0
1000
2000
3000
4000
5000
6000
0
200
400
600
800
1000
1200
1400


egreedy5
e=None
e=0.2
e=0.8
regret
trial
0
1000
2000
3000
4000
5000
0
200
400
600
800
1000
1200
1400



عملکرد الگوریتم با مقادیر epsilon بالا به صورت زیر است.

```
eps:  None  best path:  [0, 1, 5, 9, 12]  mean latency: 18.760595506353027  actions:  9000
eps:  0.2  best path:  [0, 1, 5, 9, 12]  mean latency: 18.785485644864153  actions:  9000
eps:  0.8  best path:  [0, 3, 5, 9, 12]  mean latency: 18.43289619229058  actions:  9000
```

در حالت epsilon برابر ۰.۲ تاخیر تخمینی بیشتر از حالت none است که دلیل این موضوع می‌تواند کسب نکردن داده‌های کامل باشد. چون احتمال انتخاب راندم در آن بیشتر است. در epsilon برابر ۰.۸ به صورت راندوم مسیر انتخاب شده است و به همین از گره ۳ استفاده شده که احتمال بوجود آمدن تاخیر در آن از گره ۱ بیشتر است.

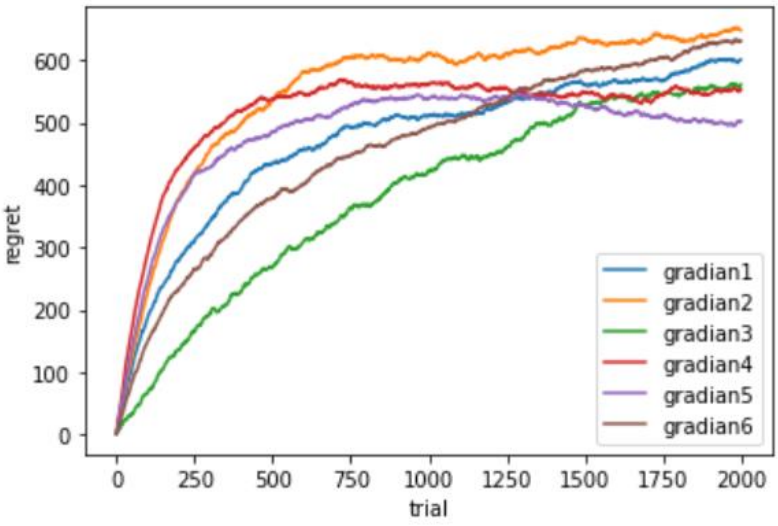
پ) در این جا الگوریتم gradient method را پیاده‌سازیم کردیم. در این الگوریتم احتمال انتخاب هر اکشن به صورت زیر بدست می‌آید.

$$\pi(a_i) = p(a_i) = \frac{e^{H(a_i)}}{\sum_b e^{H(b)}}$$

در این عبارت، تابع H نشان دهنده‌ی ترجیح برای اکشن است. در این الگوریتم می‌خواهیم با گرادیان گرفتن بر اساس این تابع و حرکت در جهت آن، مقدار مجموع پاداش‌های دریافتی را بیشینه کنیم. به روز رسانی ترجیح به صورت زیر انجام می‌شود. فرض کنیم اکشن A_t را انجام داده‌ایم و سایر اکشن‌ها را a می‌نامیم.

$$H(A_t) \coloneqq H(A_t) + \alpha \times (r - \bar{R}) \times (1 - \pi(A_t))$$
$$H(a) \coloneqq H(a) - \alpha \times (r - \bar{R}) \times \pi(a)$$

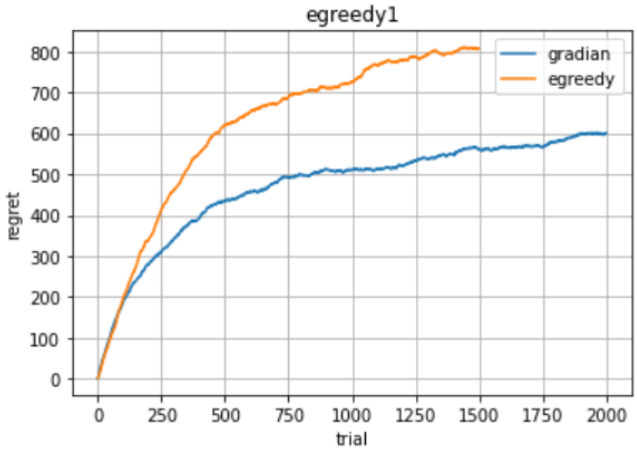
در این عبارات، $\bar{R}$ میانگین پاداش‌ها تاکنون است و r پاداش دریافتی و α نرخ یادگیری می‌باشد. در این جا اگر پاداش دریافتی به ازای اکشن انجام شده از میانگین پاداش‌ها بیشتر باشد، احتمال انتخاب آینده اکشن را با توجه به فاصله از greedy افزایش می‌دهیم. همچنین احتمال انتخاب دیگر اکشن‌ها را کم می‌کنیم که با توجه به احتمال قبلی آنها است. اگر پاداش از میانگین کمتر باشد، احتمال انتخاب آینده اکشن انجام شده را کاهش می‌دهیم. اما این کاهش به خاطر پاداش لحظه ای زیاد نخواهد بود. در این الگوریتم در ابتدا به exploration می‌پردازیم و بعد از به دست آمدن داده‌های کافی، به exploitation خواهیم پرداخت اما در این روش مانند epsilon greedy کور از فاز explore به greedy نمی‌رویم. در این جا هدفمان را بیشینه کرد جمع پاداش‌ها قرار دادیم. این حرکت در جهت این بیشینه کردن باعث می‌شود به موقع تغییر فاز دهیم. در زیر نمودار regret زیر مسئله‌ها حاصل از حل با این روش آمده است.

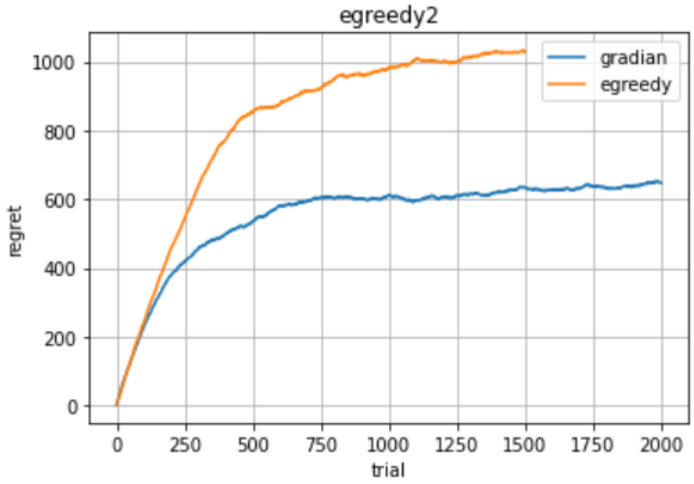


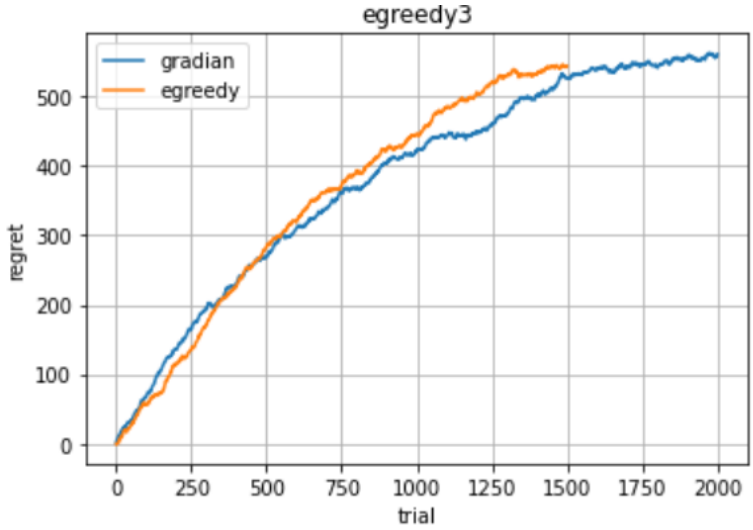
برای یادگیری از ۵۰ اجرای ۲۰۰۰ آزمایشی استفاده کردیم. همانطور که در نمودار بالا دیده می‌شود تعدادی از مسئله‌ها به ۲۰۰۰ آزمایش برای پیداکردن اکشن بهینه نیاز دارند. تعداد اکشن‌های مورد نیاز را در این سوال برابر بدترین حالت قرار می‌دهیم. یعنی اگر تعدادی زیر مسئله به جای ۱۵۰۰ اکشن با ۲۰۰۰ اکشن بتوانند اکشن بهینه را انتخاب کنند، ما تعداد اکشن مورد نیاز برای هر زیر مسئله را ۲۰۰۰ را اعلام می کنیم. نتایج یادگیری به شکل زیر است.

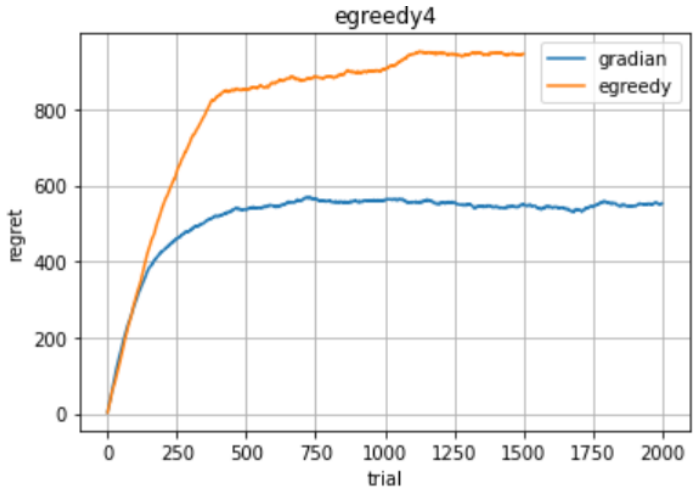
```
best path:  [0, 1, 5, 9, 12]  mean latency: 18.73171717390435  actions:  12000
```

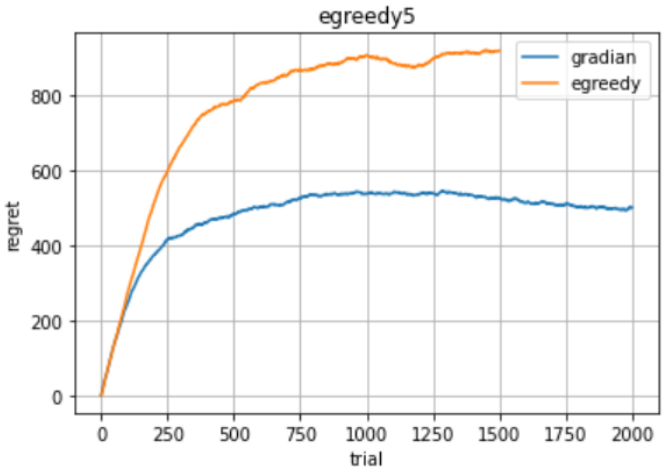
مسیر بهینه همان مسیر epsilon greedy است و تاخیر بدست آمده در صدم ثانیه با epsilon greedy فرق دارد. همان طور که دیده می‌شود تعداد اکشن‌های لازم برای یادگیری بیشتر از حالت قبل است ولی در این روش ما متوسط مجموع تاخیر کمتری در مسیر یادگیری داریم. زیر مسئله‌ها به صورت جدا در زیر مقایسه شده‌اند.

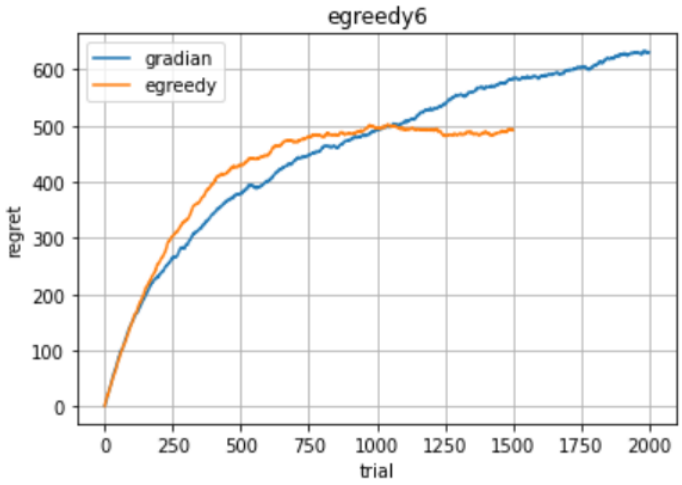

egreedy1
gradian
egreedy
regret
trial
0
100
200
300
400
500
600
700
800
0
250
500
750
1000
1250
1500
1750
2000


egreedy2
gradian
egreedy
regret
trial
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750
2000


egreedy3
gradian
egreedy
regret
trial
0
100
200
300
400
500
0
250
500
750
1000
1250
1500
1750
2000

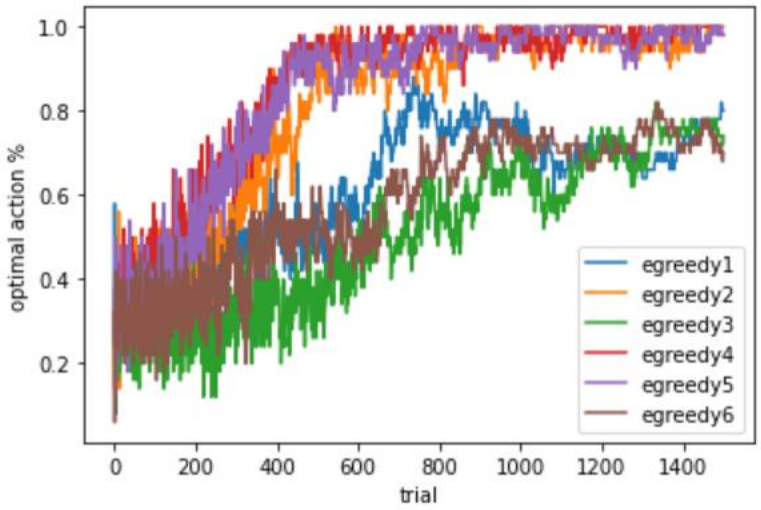


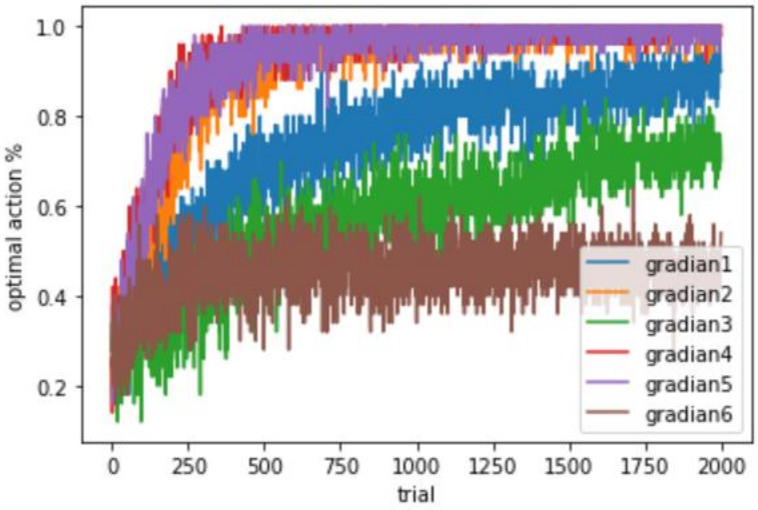




با بررسی هر زیرمسئله به صورت جدا در این دو روش می‌بینیم که تعداد اکشن مورد نیاز واقعی در روش گرادیان تفاوت چندانی با تعداد اکشن epsilon greedy ندارد و در کل با ۹۰۰۰ اکشن می‌توان به نتیجه بهینه رسید ولی تعداد بیشینه برای اطمینان کامل از حل درست تمام زیر مسئله‌ها برابر ۱۲۰۰۰ اکشن است. مورد دیگری که باید مورد توجه قرار گیرد، این موضوع است که روش گرادیان سریع‌تر اکشن

بهینه در هر زیر مسئله را پیدا می‌کند و از آن به بعد greedy عمل می‌کند و به همین دلیل regret و اندازه‌ی متوسط utility کمتر یا مساوی در تمام زیر مسئله‌ها با egreedy دارد. در این مسئله utility منفی است و هرچه اندازه ی آن کمتر باشد بهتر است. در زیر نمودار در صد استفاده از اکشن بهینه برای این دو روش آمده است که نشان دهنده‌ی سرعت بیشتر گرادیان در پیدا کردن اکشن بهینه است.

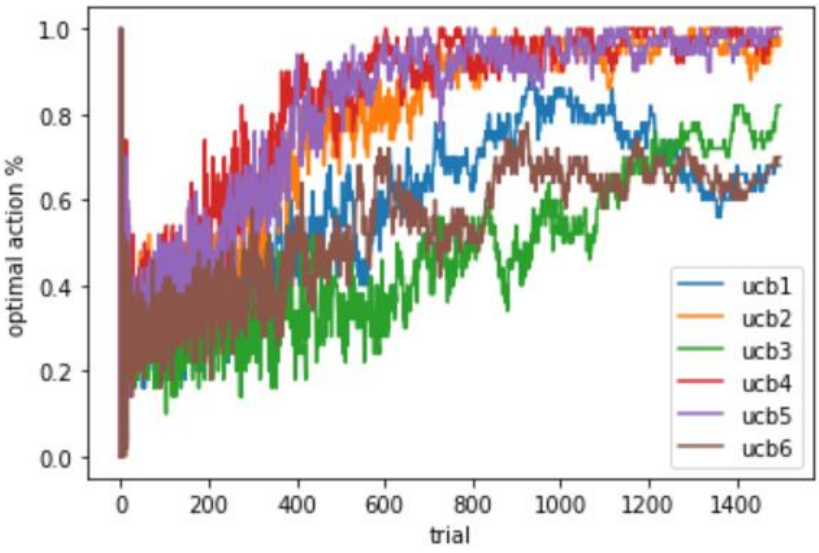


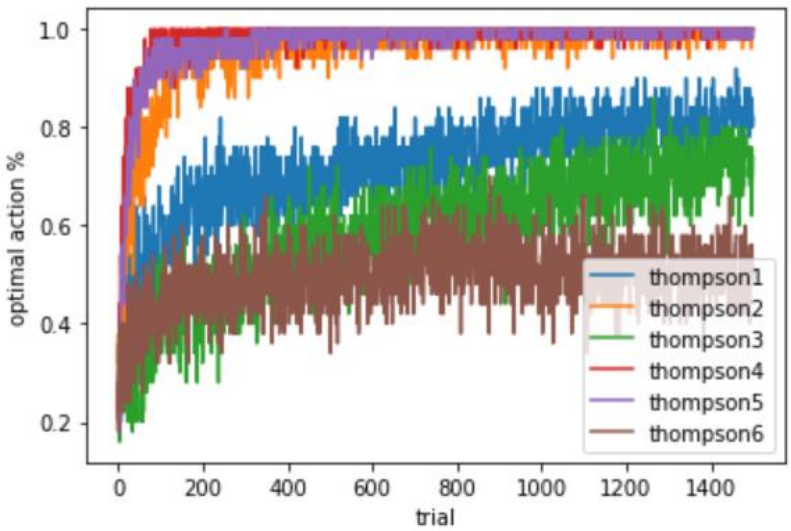


دلیل سریع تر بودن روش گرادیان، کورکورانه عمل نکردن و حل مسئله بهینه سازی برای بیشینه کردن جمع پاداش‌های دریافتی است. با توجه به این بررسی‌ها می‌توان گفت عملکرد روش گرادیان بهتر از egreedy است.

ت) بهترین روش برای این مسئله روش Thompson sampling است. در این روش توزیع پاداش هر اکشن تخمین زده‌می شود. به دلیل تخمین مناسب توزیع، ما اطلاعات بیشتری نسبت به اکشن‌ها نسبت

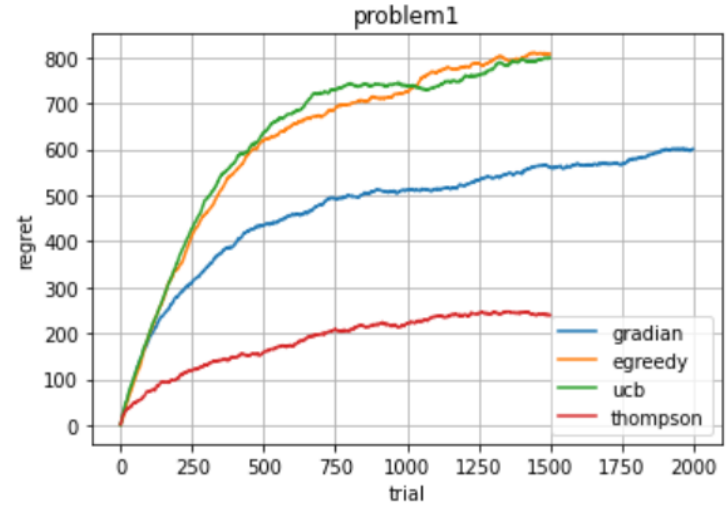
به سایر روش‌ها داریم. این موضوع باعث می‌شود که تصمیم گیری بهتری داشته باشیم. بیشترین اطلاعات ممکن زمانی بدست می‌اید که توزیع را داشته باشیم که در این روش به این سمت حرکت می‌کنیم و پارامترهای توزیع پاداش اکشن‌ها را تخمین می‌زنیم. روش دیگری که در درس بررسی کردیم، UCB است که در آن بازه‌ی اطمینان مناسب از توزیع پاداش‌ها را بدست می‌آوریم که به اندازه‌ی Thompson sampling اطلاعات در اختیار ما نمی‌گذارد. در زیر نمودار در صد استفاده از اکشن بهینه در مسائل و regret برای دو روش UCB و Thompson sampling آمده است. UCB مانند گذشته پیاده سازی شده است و برای به روز رسانی Q از نرخ یادگیری ۰.۰۲ استفاده شده است.

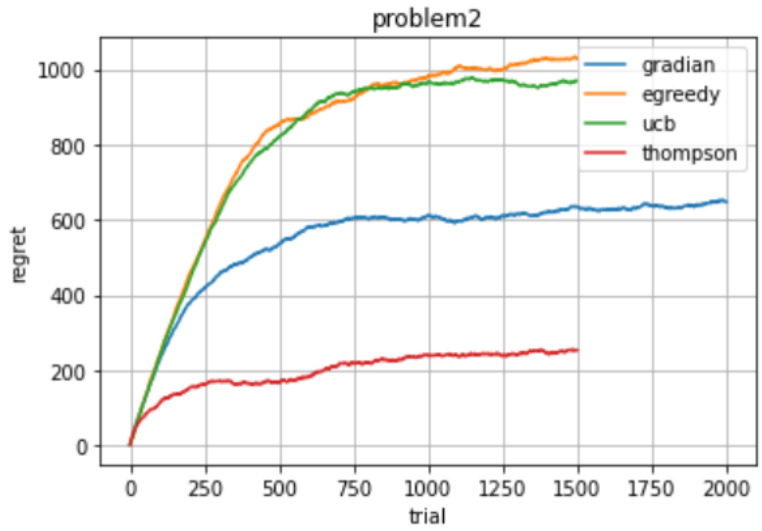


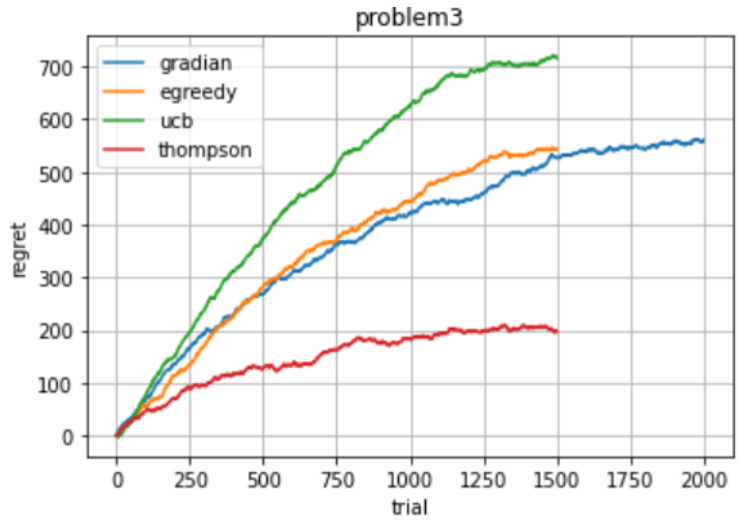


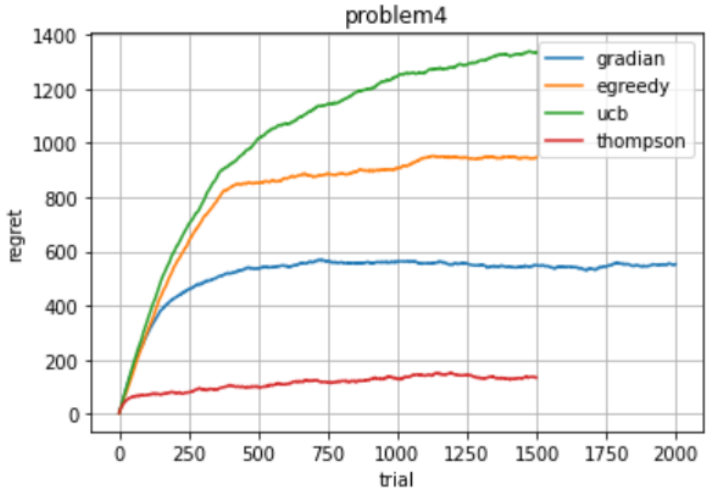
همانطور که دیده می‌شود، روش Thompson از تمام روش‌ها سریعتر اکشن بهینه را پیدا کرده و از آن به بعد با آن ادامه داده و greedy عمل کرده است. تعداد اکشن‌های لازم در این روش برای حل تمام زیر

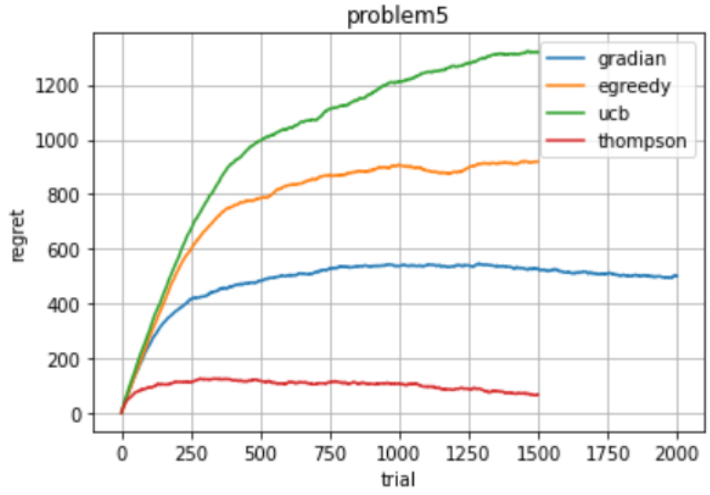
مسئله‌ها به همین دلیل از همه کمتر است. روش UCB در بدست آوردن اکشن بهینه مانند egreedy عمل کرده است. نمودار این مطالب در زیر در نمودار regret قابل مشاهده است.

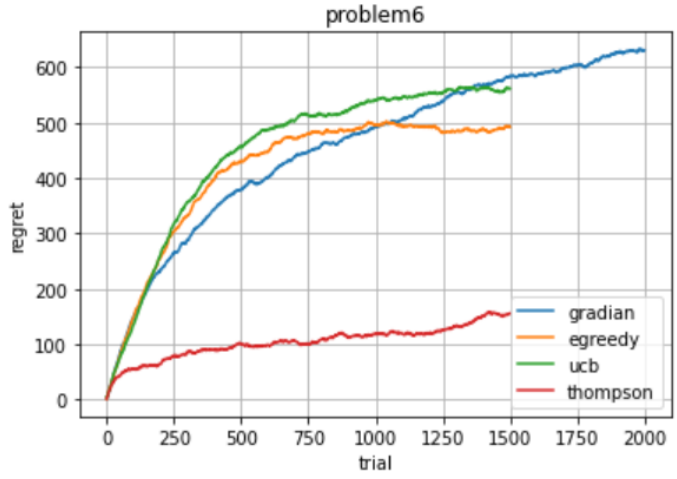












همان طور که دیده می‌شود، در تمام زیر مسئله‌ها روش Thompson در زیر ۱۵۰۰ اکشن به نتیجه رسیده است و اکشن بهینه را پیدا کرده است. به همین دلیل این روش در کمترین تعداد اکشن قادر به پیدا کردن جواب بهینه است. همچنین پیدا کردن سریع اکشن بهینه، باعث شده regret در سطح پایین

تری قرار گیرد. نتیجه یادگیریThompson و UCB همان نتایج قبلی است که به ترتیب در زیر آمده است. تاخیرها متوسط مانند قبل هستند و تفاوت صدم ثانیه دارند.

```
best path:  [0, 1, 5, 9, 12]  mean latency: 18.621562985580674  actions:  9000

best path:  [0, 1, 5, 9, 12]  mean latency: 18.619356242172724  actions:  9000
```