Deep Reinforcement Learning for Web Crawling

Konstantin Avrachenkov, Vivek Borkar, Kishor Patil

Deep Reinforcement Learning for Web Crawling

Konstantin Avrachenkov
Inria Sophia Antipolis,
Valbonne, 06902, France
k.avrachenkov@inria.fr

Vivek Borkar
Indian Institute of Technology Bombay,
Mumbai, 400076, India
borkar.vs@gmail.com

Kishor Patil
Inria Sophia Antipolis,
Valbonne, 06902, France
kishor88k@gmail.com

***Abstract*—A search engine uses a web crawler to crawl the pages from the world wide web (WWW) and aims to maintain its local cache as fresh as possible. Unfortunately, the rates at which different pages change in WWW are highly non-uniform and also, unknown in many real-life scenarios. In addition, the finite available bandwidth and possible server restrictions on crawling frequency make it very difficult for the crawler to find the optimal scheduling policy that maximises the freshness of the local cache. We model this problem in a multi-armed restless bandits framework, where each arm represents a web page or an aggregate of statistically identical web pages. The objective is to find the scheduling policy that gives the exact indices of the pages to be crawled at a particular instance. We provide an online learning scheme using deep reinforcement learning (DRL) framework which learns the unknown page change dynamics on the fly along with the optimal crawling policy. Finally, we run numerical simulations to compare our approach with state-of-the-art algorithms such as static optimisation and Thompson sampling. We observe better performance for DRL.**

***Index Terms*—Reinforcement Learning, Adaptive Web Crawling, Thompson Sampling, Multi-armed Restless Bandits.**

I. Introduction

A web crawler is used by search engines to access various web pages at certain frequencies, and depending on whether a page has changed or not, the indexing is updated in its local cache. The success of a search engine typically depends on the freshness of this update. Thus, the objective of the crawler is to find the optimal scheduling policy that maximises the freshness of the local cache. There are two major challenges in this process - i) actual change rates of pages are unknown to the crawler, ii) the crawl frequency is limited due to finite available bandwidth and possible server restrictions. One approach to obtain the near-optimal scheduling policy is to estimate actual page change rates using any of the methods provided in [6], [12], [24]. Then, these estimators can be combined with the static optimisation problem as in [8] to find the optimal crawling rates for a given page while accounting for the bandwidth constraint mentioned above. However, a major drawback for this approach is that it is an offline method, i.e., one cannot update the policy on the fly. This opens up the need for solving the on-line dynamic optimal control problem.

Some recent works [4], [18] address the same problem using a multi-armed restless bandit framework. In particular, the authors first show that the problem is Whittle indexable and then provide the exact expression for Whittle indices. These indices are then used to schedule the pages for updates, i.e., the pages having higher indices get the priority for updating over those with lower indices. It is important to note that the derivation of this *priority-index* type policy needs to know the exact dynamics of the page change rate process. In the absence of such knowledge, one can learn the Whittle indices using the Q-learning algorithm given in [5]. However, the major concern for Q-learning is that it faces a state-space explosion problem and thus, may not be able to update Q-value properly for larger state spaces. This will result in a poor estimation of Whittle indices. For other control-theoretic approaches to web crawling, we refer to [3], [16], [22].

In this paper, we consider the same problem within the restless bandit framework and propose deep learning approach for obtaining the optimal crawling policy. In particular, we use an artificial neural network to approximate the Q-value function and use it to select the page/group of pages to update at a particular time instance. Moreover, we account for the fact that the freshness of every page (or a group of pages) is not equally important to the users. For example, news pages change with very high frequency compared to other pages such as institutional or personal websites. We adjust the reward structure such that our scheduling policy accounts for how often a page changes and how important the pages are. Finally, we also provide two baseline algorithms including static optimisation (SO) and Thompson sampling (TS) and evaluate the performance of our algorithm against them.

Most of the literature in web crawling typically assumes that the page change process is Poisson point process [8]–[11]. Unlike these works, our algorithm does not require any assumption on the crawling as well as page change process. In fact, it does not even need to know the exact dynamics of the page change process.

The rest of this paper is organised as follows. The next section provides the mathematical model explaining the underlying Markov decision process (MDP) setup. In Section III, we discuss the deep reinforcement learning approach to solve the control problem. Then we discuss a couple of baseline algorithms in Section IV and present the evaluation of the performance of our algorithm against them in Section V. Finally, we conclude in Section VI with pointers to some future directions.

II. Mathematical Model

In this section, we provide the mathematical formulation for our problem. We assume that each page in the local cache has its update/change rate. The crawler needs to fetch these pages from a global directory with a certain crawling frequency so that the freshness of the local repository is optimised. Ideally, the crawler would like to access all the pages at once whenever it is scheduled to visit the global directory. However, this is often impossible due to limited available bandwidth and politeness constraint that amounts to a limitation on crawling frequency from the remote server. Thus the crawler can only update a few pages at a given time instance, which leads to the need for optimal scheduling.

The number of pages present in the local cache of the search engine typically varies from a few hundred to several thousand millions. In most real-life situations, it is very difficult, if not impossible, to track the crawling process of each page separately. One possible solution that we discuss in this paper is to group the pages having similar behaviour. This further helps to categorise the pages according to their importance and their change frequency. We then model the underlying problem as multi-armed restless bandits where each group represents an arm of the bandit. At each time instance, one group is selected from which we select a single page for crawling. Note that the number of pages in the group can be different, however, we assume that all pages inside any given group are indistinguishable. For simplicity, we chose one page at a time to crawl using a round-robin scheme from the selected group.

The dynamics of each group are governed by a Markov Decision process (MDP). The details are as follows.

State space[1]: The state of kth arm (group) is represented by $(X_k(t), Y_k(t))$ where $X_k(t) \in \mathbb{N} = 1, 2, 3, \ldots$ is the number of time slots since it was last chosen and $Y_k(t) = 1$ if the page to be selected from group k has been updated in the last visit, zero otherwise. Then $Z(t) = \big(X_k(t), Y_k(t)\big)_{k=1}^{K}$ denotes the state of the bandit process where K is the number of arms. For simplicity, we assume that $X_k(t)$ takes value in $\mathcal{X}^k$ which is finite or countable. For future reference, let N_k be the total number of pages in group k.
Actions: Each arm has a binary action process $\{0, 1\}$ depending on whether it is selected or not.

$$A_k(t) := \begin{cases} 1, & \text{if arm } k \text{ is chosen at time } t \\ 0, & \text{otherwise.} \end{cases}$$

From the chosen group, we then select one page according to the round-robin scheme. Thus, a joint control action $A(t) = \big(A_k(t)\big)_{k=1}^{K}$ denotes the decision made by the bandit at time t. Since we can only choose one arm at each time, $A(t)$ must have only one nonzero component, or equivalently,

$$\sum_{k=1}^{K} A_k(t) \leq 1, \quad \text{for } t = 0, 1, 2, \ldots \tag{1}$$

The extension to the more general case of selecting $K > 1$ arms at a time is rather straightforward, for example see [4], [5].
Rewards: The reward at each stage depends on two factors: i) the group from which a page is selected for crawling and, ii) the indicator whether the selected page from the group has been changed or not since it was last crawled.

Let μ_k represents the 'importance' of the group k. The importance of page is related to how often it is changing and the impact of its freshness on overall system. Recall that at each time instance we select one page from a chosen arm in round-robin fashion. Let us define a pointer $D_k(t)$ which returns the id of the page to be selected at time t from group k. We now define the indicator $I_k^j(t)$ as below,

$$I_k^j(t) = 1_{\{j = D_k(t)\}}\, 1_{\{\delta_j(t) = 1\}} \tag{2}$$

Here, the indicator function $1_{\{\cdot\}}$ evaluates to 1 if its argument is true and to 0 if this is not the case. Furthermore, $\delta_j(t)$ is the indicator whether page j has been modified or not at time t since it was last crawled. In simple terms, $I_k^j(t)$ is 1 if the selected page j from group k was modified since it was last crawled. We now define the state and action dependent one-slot net rewards for group k as,

$$R_k(Z_k(t), A_k(t)) = \mu_k\, A_k(t) \sum_{j=1}^{N_k} I_k^j(t) \quad \text{for } \mu_k > 0. \tag{3}$$

State Dynamics: The state of an arm evolves as follows with probability 1,

$$X_k(t+1) := \begin{cases} 0, & \text{if } A_k(t) = 1 \\ X_k(t) + 1, & \text{if } A_k(t) = 0. \end{cases} \tag{4}$$

and

$$Y_k(t+1) := \begin{cases} 1_{\{\delta_{D_k(t)}(t) = 1\}}, & \text{if } A_k(t) = 1 \\ Y_k(t), & \text{if } A_k(t) = 0. \end{cases} \tag{5}$$

Objective: For infinite horizon optimisation problem, our objective is to find a long-run discounted-optimal policy which maximises the following objective function with respect to the policy π,

$$E^{\pi}\left[\sum_{t=0}^{\infty} \sum_{k=1}^{N} \gamma^t\, R_k(Z_k^{\pi}(t), A_k^{\pi}(t))\right] \tag{6}$$

subject to,

$$\sum_{k=1}^{N} A_k(t) = 1, \quad \text{for } t = 0, 1, 2, \ldots \tag{7}$$

Here $\gamma \in (0, 1)$ is the discount factor. The above constraint ensures that no more than one group (arm) is active at

[1]We note that in this model we need to work with partially observed states. Specifically, variable Y_k becomes observable only after selecting arm k. However, this is not a problem since we can view the partial observations as a restriction on the neural network.

each time instance. Furthermore, only a single page is selected from the chosen group. This problem is in general computationally intractable [19].

III. Deep Reinforcement Learning

The Q-Learning [25] algorithm works well when the environment is small, simple and the function $Q(s, a)$ can be represented using a table or a matrix of values. The problem becomes quickly intractable as the size of the state and/or the action spaces increase. This is often referred to as the state-space explosion problem in Markov decision processes. One possible solution is to use deep neural networks to approximate the Q-function, replacing the need for a table to store the Q-values. The resulting algorithm is known as Deep Q-Networks (DQN) [17] reinforcement learning. The main challenge lies in designing the deep neural network which can approximate the Q-function appropriately.

We now present the details of the neural network that we use in this paper. The input is nothing but the state of the system that consists of two separate vectors $X(t)$ and $Y(t)$,

- Elapsed time since last crawled for each group $X(t) = \big(X_k(t)\big)_{k=1}^{K}$.
- Indicator if the page selected for crawling from a particular group was up-to-date or not during the last crawl $Y(t) = \big(Y_k(t)\big)_{k=1}^{K}$.

The output of the neural network is the set of K real values which is equal to the number of arms (groups). We select the arm with the highest value. As discussed in the earlier section, once the arm (group) is chosen, we select the page to crawl from that group sequentially. To balance exploration-exploitation trade-off, we use epsilon-greedy scheme.

We use the full gradient DQN (FG-DQN) algorithm explained in [7]. We make a slight modification in FG-DQN in order to adapt it to our problem setup. In particular, we do not fetch samples from replay memory by fixing the current state-action pair, whereby we avoid the computational effort of using special experience replay used in [7]. We now explain the motivation behind this. The input to the neural network is the state of the system which has twice the number of elements as the total number of arms. While running the simulations, we observed that it was difficult to get multiple samples from replay memory by fixing state-action pair at each time. This is because we have to find out the samples matching all the elements of state and action. We mostly observed only one sample from replay memory. Therefore we decided to avoid using the special experience replay. However, we still use a batch of random samples to update the parameters of the neural network as in the original FG-DQN. The pseudo-code for the DRL algorithm is explained in Algorithm 1. See Appendix for more details.

Algorithm 1: DRL - Web crawling

Input: replay memory $\mathcal{D}$ of size M, minibatch size B, number of Iterations T, discount factor γ and, exploration probability ϵ.
Initialise the weights θ randomly for the Q-Network.
Initialise pointers M_k to zero so that they denote first page of each group.
Receive initial observation $s_1 = \{X_0, Y_0\}$.
for *n = 1 to T* **do**
  Generate a uniform random variable U_n with support [0,1]
end
if $U_n < \epsilon$ *(here, Uni[0,1] is uniform random)* **then**
  Select the arm U_n at random.
else
  $U_n = Argmax_u Q(Z_n, u; \theta)$
end
Select the page in the corresponding group and fetch it.
If the pointer denotes the last page of the group then reset M_k to 0 otherwise increase M_k by 1 to denote next page in the group.
Observe the reward R_n and obtain next state Z_{n+1}.
Store the tuple (Z_n, U_n, R_n, Z_{n+1}) in $\mathcal{D}$.
Sample random minibatch of B tuples from $\mathcal{D}$.
Compute gradients and using [7, Eq.(19)] without considering an experience replay.
update parameters θ.

IV. Baseline Algorithms

A naive algorithm is to select an arm uniformly at random. We will observe in the numerical section that such an approach can be quite wasteful even for a small number of arms and can fail completely for complex problems. For a fair comparison, we provide two different baseline algorithms which efficiently balance the exploration-exploitation trade-off.

A. Thompson Sampling (TS)

TS [23] was originally introduced for clinical trials in 1933 to allocate experimental effort in two-armed bandit problems and was later adapted widely for many applications, e.g., [1], [13], [14]. We adopt here a similar approach by considering Bernoulli bandits. Moreover, we work with Beta distributions because of their conjugacy properties. In particular, we start with parameters $(\alpha_k = 1, \beta_k = 1)$ for all arms and then update these parameters using the Bayesian update rule i.e., α_k or β_k increases by one with each observed success or failure, respectively. Note that only the parameters of the selected arm are updated. At each time instance, random samples are generated from these Beta distributions and the arm corresponding to the maximum value of this sample is selected. We get a reward equal to the importance of the arm only if the selected page has been changed since its last visit. For pseudo-code, we refer to [2, Algorithm 2].

B. Static Optimisation (SO)

The goal of the server is to maximise the average freshness given by,

$$E\left[\frac{1}{T}\sum_{t=1}^{T}\sum_{k=1}^{K}\mu_k 1_{\{Fresh(k,t)\}}\right]$$

where T is the time horizon and $Fresh(k,t)$ is the event that page k is fresh at time t. For large T, Azar et al. [8] showed that the above objective function corresponds to following:

$$F(p) = \sum_{k} \frac{\mu_k\, p_k}{p_k + \Delta_k - \Delta_k\, p_k}. \tag{8}$$

Here Δ_k is the probability that page k is changed in a given time unit, independently of other pages and time steps. The update policy is denoted by a distribution $\{p_i\}_{i=1}^{k}$ over pages. We assume that only one page will be crawled at a time and obtain the optimal policy (p_i^*) using [8, Algorithm 1]. Note that, the problem set-up assumes that the server has full knowledge of the actual change rate process of all the pages.

To implement the static optimisation policy, we first obtain the optimal probability distribution assuming the full knowledge of change rate probabilities of each page. Then we consider the multi-arm bandit framework where we select an arm according to this distribution at each time instance. The reward structure is the same as in the above cases.

V. Numerical Examples

In this section, we present numerical simulations to compare the performance of our approach to baseline algorithms. For simplicity, we assume that all the pages can be grouped into four different categories as described below. The percentage next to the category represents the fraction of volume each group occupies from the local cache.

1) Small, expensive and frequently changing (5%)
2) Large, inexpensive and infrequently changing (80%)
3) Moderately large, expensive and infrequently changing (10%)
4) Small, inexpensive and frequently changing (5%)

Let λ_i, $i \in \{1,2,3,4\}$ represents the actual rates at which each page changes in group i and let N_i be the number of pages in group i. Note that these rate should satisfy [2] following condition, $\sum_{i=1}^{4} N_i\, \lambda_i \leq 1$

We start with a simple case, i.e., assuming one page per group. Thus each arm in the bandit framework represents a single page. The change rates are fixed at $[0.6, 0.08, 0.01, 0.3]$. Note that these rates are only used to generate an observation if a particular page has been changed in the particular time slot or not. As a result, the system can only detect the change in the page if any, but has no knowledge of how many times a page has been updated in a given time slot. We further make the natural assumption that each page has different importance μ_k and set $\mu = [5, 0.2, 1, 0.5]$.

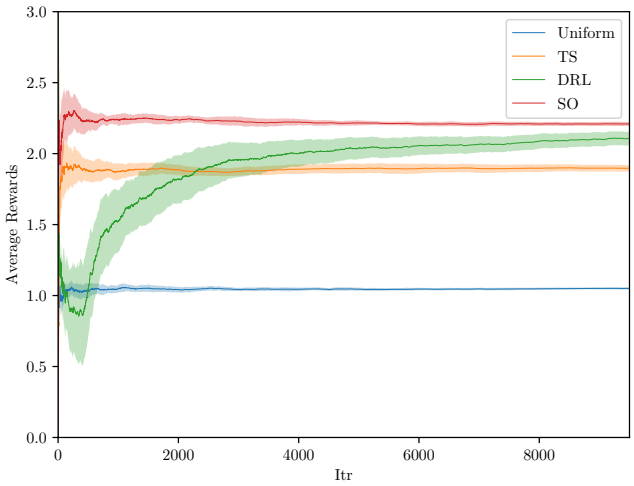

Fig. 1. Running Average: Single page in each group

We use a neural network with two hidden layers. The number of neurons for the hidden layers are 4000, and 8 respectively. For non-linearity, we use ReLU activation after each hidden layer. we use the on-policy version with the popular 'ϵ-greedy' scheme which picks the current guess for the optimal choice with probability 1 - ϵ and chooses a control uniformly with probability ϵ. We start with $\epsilon = 0.7$ and slowly decrease it with the iteration to $\epsilon = 0.1$. The discount factor for the MDP is set to $\gamma = 0.95$.

At each iteration, we save the immediate reward and then calculate the running average. We run the experiment for 10 runs and plot the average in Fig. 1. The shaded region denotes the 95% confidence interval for the averaged reward. It is easy to verify from the Fig. that Thompson sampling, DRL and static optimisation (SO) outperform the uniform policy. Furthermore, the average reward for DRL and SO is higher than that of TS. We also observe that the variance for DRL is higher initially but is reduced with number of iterations.

In case of SO, the optimal probability distribution is $[0.942, 0.055, 0.003, 0.]$. We select one arm according to this distribution at each time. Since this is a threshold policy, it may happen that some pages have zero probability of being selected. For example, in this example, page 4 is never crawled. This is unfavourable in many real-life scenarios and is another limitation of SO policy in addition to the need of knowing λs beforehand. On the other hand, TS is an online learning scheme that starts with an equal chance of selecting an arm and then improves the parameter of each page depending on the success or failure of detecting the page change. It balances the exploration-exploitation trade-off efficiently. Intuitively, the page with the highest number of changes will be selected most often given all else equal. However, if this page is selected too often then the system will fail to detect the changes and thus decreases the probability of being chosen. In the long run, TS will try

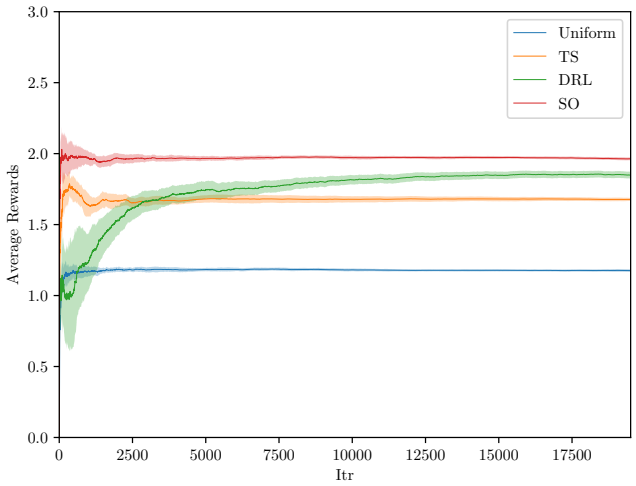

Fig. 2. Running Average: Multiple pages in each group

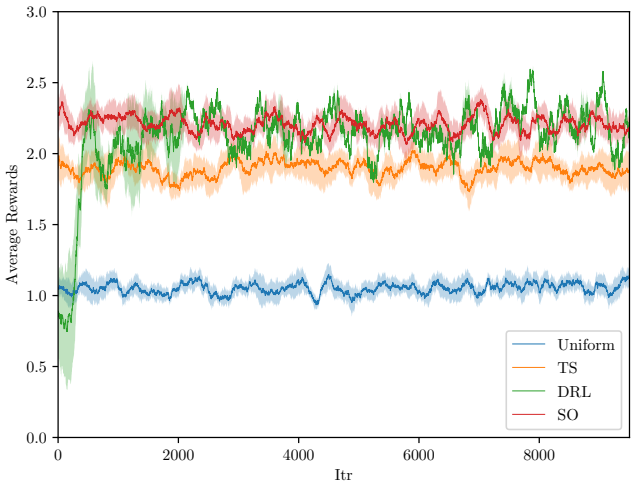

Fig. 3. Moving Average: Single page in each group

to learn the actual change rate of the pages.

We now move on to a more general case. In practice, several web pages having the same properties can be grouped such as news pages, educational websites etc. Note that one can use different features to group the pages. The only assumption we make in the group setting is that the pages inside the groups are indistinguishable and behaves similarly. Next, we identify each arm of the bandit with one group. When an arm is selected at a particular time, we crawl the pages inside the group sequentially. More precisely, we maintain a pointer M_k for each group. When group k is selected, we crawl the page denoted by pointer M_k and increase it by 1. Thus, the next page gets crawled when the same group is selected next time. For the non-selected group, M_k remains the same. Lastly, if all the pages are crawled in the group, the pointer is set to 0 to denote the first page in the group.

For this simulations, we fix the number of pages per group as $[5, 80, 10, 5]$ so that the total number of pages in local cache of the server is 100. We choose the change rates to be $\lambda = [0.1, 0.002, 0.005, 0.05]$. This is done to suit to four groups mentioned earlier i.e., $\lambda_1 > \lambda_4 >> \lambda_3 > \lambda_2$. Note that one can make different groups according to need of the application and set $\lambda_i's$ accordingly.

We use the same neural network and the discount factor is again set to 0.95. Fig. 2 depicts the running average for each algorithm. We observe similar behaviour as in the previous experiment.

We further observe from Fig. 1 and Fig. 2 that DRL may take some time to learn the optimal policy. This may be due to two reasons. First, training the parameters of the neural network used to approximate Q-value may take several iterations. Second, the running average penalises initial reward heavily and thus affecting the overall trajectory. To observe latter behaviour closely, we plot the moving average over a window of 200 iterations for both examples in Fig. 3 and Fig. 4. The performance behaviour is still the same, but with high fluctuations across all policies. We confirm from these figures that DRL indeed will show better performance over TS in 500 iterations instead of 3000 iterations observed in running average plots. Since TS is still very quick, one general recommendation is to use TS for the initial few iterations and then switch to DRL.

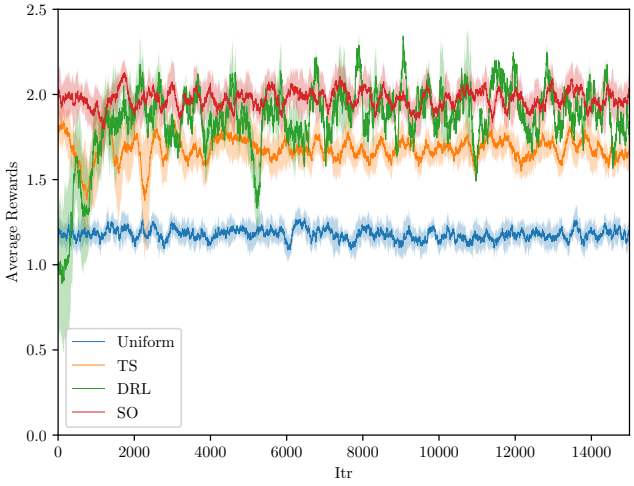

Fig. 4. Moving Average: Multiple pages in each group

We end our discussion by pointing out major benefits of DRL over SO. The SO may have slightly better performance than DRL, but it may be unfavourable from practical point of view. This is because SO is a threshold type policy, i.e., it selects an arm with probability distribution which may contain several zeros. Thus, some groups are never crawled, which means that all the pages in those group will not be visited ever. Such behaviour will then allow exposure to only

selective groups forming bubbles, see [20], [21]. Another major drawback for SO is that it requires full knowledge of λs and has a strong assumption that crawling process is a Poisson point process. Our approach does not suffer from any of these problems and has no assumption on the crawling process.

VI. Conclusion

We have introduced deep reinforcement learning for adaptive web crawling, in particular, the use of DQN for multi-arm restless bandits. We observe better performance of DRL than baseline algorithms in numerical examples. For a future direction, it will be interesting to compare the performance of the above algorithms on real data sets.

VII. Acknowledgement

The work of VSB was supported in part by an S. S. Bhatnagar Fellowship from the Council of Scientific and Industrial Research, Government of India. The work of KP and KA is partly supported by Projet PIA - ANSWER - FSN2 (P159564-2661789\DOS0060094) and the project of Inria - Nokia Bell Labs "Distributed Learning and Control for Network Analysis". This work is also partly supported by the project IFC/DST-Inria-2016-01/448 "Machine Learning for Network Analytics".

Appendix

Full Gradient DQN (FG-DQN):

FG-DQN is a modification of DQN proposed in [7] which provides better theoretical guarantees over traditional DQN or Double DQN. In particular, it uses artificial neural network to approximate the Q-function and update the parameter θ of this neural network with following update scheme,

$$\theta_{n+1} = \theta_n - a(n)\times \left(\overline{(r(Z_n,U_n) + \gamma \max_v Q(Z_{n+1},v;\theta_n) - Q(Z_n,U_n;\theta_n))}\times (\gamma \nabla_\theta Q(Z_{n+1},v_n;\theta_n) - \nabla_\theta Q(Z_n,U_n;\theta_n)) + \xi_{n+1} \right) \quad (9)$$

Here, (Z_n, U_n) is the state action pair at time n and $r(\cdot)$ represents the immediate reward for current state-action pair. γ is the discount factor and ξ_{n+1} is extraneous i.i.d. noise componentwise distributed independently and uniformly on $[-1, 1]$. The overline stands for a modified form of experience replay which comprises of averaging at time n over past traces sampled from $(Z_k, U_k, Z_{k+1}), k \leq n$, for which $Z_k = Z_n, U_k = U_n$.

The important thing here is to note that the sequence $\{\theta_n\}$ generated by FG-DQN (9) is a stochastic (sub-)gradient scheme for the true Bellman error given by,

$$\bar{\mathcal{E}}(\theta) = E\left[\left(r(Z_n,U_n) + \gamma \sum\nolimits_y p(y|Z_n,U_n)\times \max_v Q(y,v;\theta_n) - Q(Z_n,U_n;\theta) \right)^2 \right]$$

References

[1] Agrawal, S. and Goyal, N., 2013. Thompson sampling for contextual bandits with linear payoffs. ICML 2013, 127-135.

[2] Andrews, M., Borst, S., Lee, J., Martin-Lopez, E. and Palyutina, K., 2020. Tracking the State of Large Dynamic Networks via Reinforcement Learning. IEEE INFOCOM 2020, 416-425.

[3] Avrachenkov, K., Dudin, A., Klimenok, V., Nain, P. and Semenova, O., 2011. Optimal threshold control by the robots of web search engines with obsolescence of documents. *Computer Networks*, 55(8), 1880-1893.

[4] Avrachenkov, K.E. and Borkar, V.S., 2016. Whittle index policy for crawling ephemeral content. IEEE TCNS, 5(1), 446-455.

[5] Avrachenkov, K., and Borkar, V.S., 2020. Whittle index based Q-learning for restless bandits with average reward. arXiv:2004.14427.

[6] Avrachenkov, K., Patil, K. and Thoppe, G., 2020. Change rate estimation and optimal freshness in web page crawling. ValueTools 2020.

[7] Avrachenkov K.E., Borkar V.S., Dolhare H.P., Patil K., 2021. Full gradient DQN reinforcement learning: A provably convergent scheme. In: Piunovskiy A., Zhang Y. (eds) *Modern Trends in Controlled Stochastic Processes*, 192-220. Springer.

[8] Azar, Y., Horvitz, E., Lubetzky, E., Peres, Y. and Shahaf, D., 2018. Tractable near-optimal policies for crawling. PNAS, 115(32), 8099-8103.

[9] Brewington, B.E. and Cybenko, G., 2000. How dynamic is the web? *Computer Networks*, 33(1-6), 257–276.

[10] Brewington, B.E. and Cybenko, G., 2000. Keeping up with the changing web. *Computer*, 33(5), 52–58.

[11] Cho, J. and Garcia-Molina, H. 2000. The evolution of the web and implications for an incremental crawler. VLDB 2000, 1–18.

[12] Cho, J. and Garcia-Molina, H., 2003. Estimating frequency of change. ACM TOIT, 3(3), 256-290.

[13] Hill, D.N., Nassif, H., Liu, Y., Iyer, A. and Vishwanathan, S.V.N., 2017. An efficient bandit algorithm for realtime multivariate optimization. ACM SIGKDD, 1813-1821.

[14] Kawale, J., Bui, H.H., Kveton, B., Tran-Thanh, L. and Chawla, S., 2015. Efficient Thompson sampling for online matrix-factorization recommendation. NIPS 2015, 1297-1305.

[15] Lin, L.J., 1993. Scaling up reinforcement learning for robot control. ICML 1993.

[16] Liu, Z. and Nain, P., 2006. Optimization issues in Web search Engines. *Handbook of Optimization in Telecommunications*, 981-1015. Springer.

[17] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. and Petersen, S., 2016. Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.

[18] Nino-Mora, J., 2014. A dynamic page-refresh index policy for web crawlers. ASMTA 2014, 46-60.

[19] Papadimitriou, C.H. and Tsitsiklis, J.N., 1999. The complexity of optimal queuing network control. *MOR*, 24(2), 293-305.

[20] Pariser, E., 2011. The filter bubble: What the Internet is hiding from you. Penguin.

[21] Resnick, P., Garrett, R.K., Kriplean, T., Munson, S.A. and Stroud, N.J., 2013. Bursting your (filter) bubble: strategies for promoting diverse exposure. CSCW 2013, 95-100.

[22] Talim, J., Liu, Z., Nain, P. and Coffman Jr, E.G., 2001. Controlling the robots of Web search engines. ACM Sigmetrics 2001, 236-244.

[23] Thompson, W.R., 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4), 285-294.

[24] Upadhyay, U., Busa-Fekete, R., Kotlowski, W., Pal, D. and Szorenyi, B., 2020. Learning to crawl. AAAI 2020, 6046-6053.

[25] Watkins, C.J. and Dayan, P., 1992. Q-learning. *Machine Learning*, 8(3-4), 279-292.