Development of news article recommendation system via reinforcement learning.

(Parallelization of bandit algorithms to reduce computational cost of news article recommendation systems.)

AISTARS
AICP 2021

Abstract
Impacts on performance of bandit algorithms in recommending news article when adapted for recommending to multiple users in parallel is studied in this work. The algorithms explored in this work are sequential LinUCB, sequential Thompson Sampling, parallelized Lazy LinUCB, parallelized Lazy Thompson Sampling, parallelized non-Lazy LinUCB, parallelized non-Lazy Thompson Sampling. The algorithms belong to a class of contextual bandit algorithms which in turn is a subclass of multiarmed bandit algorithms. Aim in this work is to explore whether parallelized implementation for faster performance at the expected sacrifice of rewards is worth using in a practical scenario – the scenario being news article recommendation to users.

Introduction
Multiarmed bandit problem refers to a scenario where a hypothetical agent has access to a limited set of choices, where each of the choices yield a reward unknown to the agent. The only way for the agent to get an estimate of the underlying reward distribution is to first try out a few choices and to observe the reward obtained for the corresponding choices. Different bandit algorithms offer different theoretical basis on which to estimate the long-term expected reward of each of the choices or arms based on the reward/trial history. The agent then uses the current estimate to choose arms to maximize the long-term reward gain from limited tryouts/trials at hand.
The algorithms we have implemented are **contextual** bandit algorithms.
At every time step, the algorithms use the information about:v
-the pool of available news articles, each referred to as an "arm", a in the literature.
- **the yielded reward r_t** (equals "1"if a user clicks on the recommended article or "0" otherwise) for each of those past trials ,
- **the context vector $x_{t,a}$** for each of those trials to estimate the reward payoff function at the next tim e step. Context vector, **$x_{t,a}$** can be said to be the vector representation of the user-article scenario at a particular time step as shown in **Figure 1**.
- tryout count for each arms. This is done by updating the covariance matrix as $V_t \leftarrow V_{t-1} + x_{t,a_t}^T x_{t,a_t}$ (news articles which were chosen and then recommended to the users),

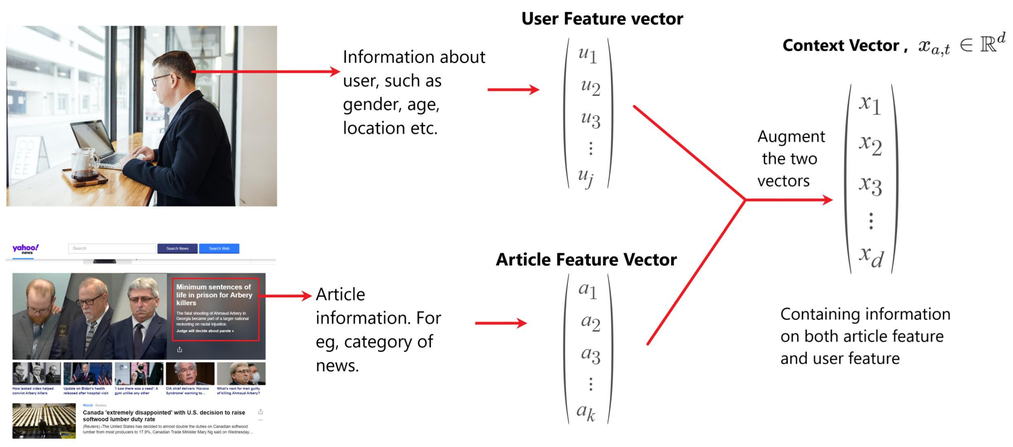

Figure 1. *Context vector derivation.*

The expected reward function for all the algorithms given the context vector is assumed to be linear. Expressed as: $\mathrm{E}[r_t|x_{t,a}] = x_{t,a}^T\,\theta$ where θ is iteratively estimated as $\hat{\theta}$ at each time step after the rewar d is observed.

Sequential algorithms vs parallel algorithms
In the two sequential algorithms, LinUCB and Thompson sampling, the agent uses a single worker which attends to a single user in the queue at a particular time step. On observing the reward the worker calculates the estimate of the expected reward for each of the article to be displayed, **a_t** given the context. The method with which this expectation is calculated is nuanced for different bandit algorithms.
LinUCB calculates the vector b_t by iterative update $b_t \leftarrow b_{t-1} + x_{t,a_t} r_t$. And then at time t + 1, $\theta_{t+1} \leftarrow V_t^{-1} b_t$ is used to estimate max expected reward for each arm as :
$\theta_{t+1}^T x_{t+1,a} + \alpha\sqrt{x_{t+1,a}^T V_t^{-1} x_{t+1,a}}$ where α is a hyperparameter that encourages higher selection frequency for underexplored arms. The arm/article with the highest reward estimate is then chosen to be recommended.
In case of Thompson Sampling algorithm, all parameter updates are the same except at the two following points. While $\widehat{\theta_{t+1}} \leftarrow V_{t+1}^{-1} b_t$, the estimate is then used as mean for the normal distribution $\mathrm{N}(\widehat{\theta_{t+1}}, \nu^2 V_{t+1}^{-1})$, which is then sampled from to get an estimate for θ to be then used to compute maximum reward estimate for each arm as $\widehat{\theta_{t+1}}^T x_{t+1,a}$before an arm is selected. Upon observing the rewards and updating all the relevant parameters , the worker receives the next user in queue for the next iteration
The sequential algorithms keep a pseudocount of features encountered in previous time steps using a covariance matrix V_t, whose value at time t is computed as:
$\mathbf{V_t \leftarrow \lambda I_d + \sum_{t=1}^{t-1} x_{t,a_t}^T x_{t,a_t}}$ (λ being a regularization term) or iteratively as $V_t \leftarrow V_{t-1} + x_{t,a_t}^T x_{t,a_t}$. The sole worker in sequential algorithms uses the entire history of encountered context vectors through the covariance matrix and also the pseudocount of rewards for corresponding vectors via a parameter b_t (computed as $b_t \leftarrow b_{t-1} + x_{t,a_t} r_t$) when choosing a news article for a particular user.
However, real life scenario demands that multiple users be served simultaneously. Hence, the need for the sequential algorithms adapted to be used for the scenario involving **P** parallel workers serving simultaneously arises (**Figure 3**).

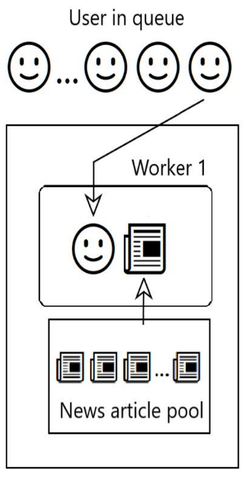

Figure 2. *Workers in sequential bandit algorithms*

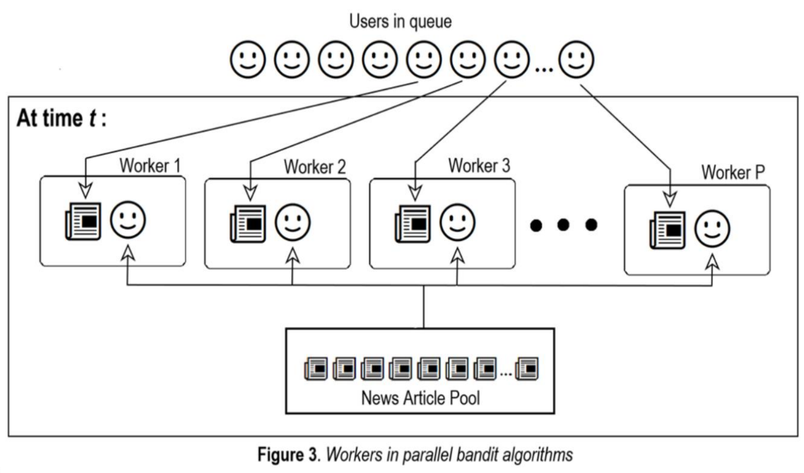

Figure 3. *Workers in parallel bandit algorithms*

The parallel version for both the LinUCB and Thompson Sampling has **non-lazy** and **lazy** variants. The parallel bandit algorithms serve P users in batch at time t. In case of non-lazy versions, covariance matrix $V_{t,p}$ is updated at the end of an iteration involving P workers, so in the following iteration, the whole batch of workers use the same $V_{t,p}$ updated at the end of previous iteration. In lazy version, the covariance matrix is updated within the batch by adding outer product of context vectors encountered by the previous workers within the same batch. The covariance is as follows:
$\mathrm{V_{t,p}} = \lambda \mathrm{I_d} + \sum_{e=1}^{t-1} \sum_{f=1}^{P} \mathrm{x_{e,f,a}\, x_{e,f}^T} + \sum_{k=1}^{p-1} \mathrm{x_{t,k,a}\, x_{t,k,a}^T}$
The rewards for individual workers in both versions are observed only after arm selection for the entire batch has been done. So, $\hat{\theta}_t$ update takes place only after rewards after entire batch has been calculated.
The key disadvantage in both parallel versions is that workers are not aware of the reward yield of the workers in the same batch. While the workers in lazy versions are aware of the arm selections up to the previous worker, through the covariance matrix, the workers in non-lazy version are only aware of the arm selections of the workers up to the last batch in previous time step. These issues make parallel algorithms take "less informed" decisions than their sequential counterpart, which have access to both the reward yield and the arm selections up to the previous served user. The relative "extra" information about arm selection in Lazy versions of the algorithm induces more variable arm selection compared to non-lazy ones. This leads the belief that lazy versions would accumulate more clicks/rewards in the experiment compared to non-lazy versions.

Experiment Procedure
Dataset used: Yahoo! Front Page Today Module dataset R6A.
Processor Used: Intel Core i7-10700KF 5.1 GHz
Dataset contents:
6. user features, 6 Article features, ID of the article being displayed to the user
observed reward for the article shown, pool of article ID's at time t from which article to be displayed is chosen.

Performance measure: $\mathrm{Click\_Through\_Rate}(CTR) = \frac{Total\_Cumulative\_Clicks}{Total\_Cumulative\_Users\_Served}$

Experiment description:

- Hyperparameters and parameters are first initialized. At each iteration, user features and article features are used to form the context vector. If the selected article in experiment matches the selected arm in mentioned in the dataset, the rewards for that context is observed. All the parallel algorithms have been merged and shown below . Setting p = 1 gives the sequential version of the algorithms:
  - $V_{t,p} = \lambda I_d$, $b_t = 0_d$, set algorithm {LinUCB ,Thompson Sampling}, set α,set version{Lazy,Non-Lazy}
  - For t = 1,2,3,…,T do
    - $\widehat{\theta_t} \leftarrow V_t^{-1} b_{t-1}$
    - For p = 1,2,3…P
    - If version is Lazy:
      - Compute $\mathrm{V_{t,p}} = \mathrm{V_{t-1,1}} + \sum_{k=1}^{p-1} x_{t,k,a_t}^T\, x_{t,k,a_t}$
    - For articles a = 1,2,3…K do
      - If algorithm is LinUCB :
        - If version is Lazy:
          - Reward expectation for $a = \widehat{\theta_t^T} x_{t,p,a} + \alpha\sqrt{x_{t,p,a}^T V_{t,p-1}^{-1} x_{t,p,a}}$
        - If version is Non-Lazy:
          - Reward expectation for $a = \widehat{\theta_t^T} x_{t,p,a} + \alpha\sqrt{x_{t,p,a}^T V_{t,1}^{-1} x_{t,p,a}}$
      - If algorithm is Thompson Sampling:
        - If version is Lazy:
          - Reassign $\widehat{\theta_t} \leftarrow Sample\ from\ N(\widehat{\theta_t}, \nu^2 V_{p-1,1}^{-1})$
          - Reward expectation for $a = \widehat{\theta_t^T} x_{t+1,p,a}$
        - If version is Non-Lazy:
          - Reassign $\widehat{\theta_t} \leftarrow Sample\ from\ N(\widehat{\theta_t}, \nu^2 V_{t,1}^{-1})$
          - Reward expectation for $a = \widehat{\theta_t^T} x_{t,p,a}$
      - End the loop for articles
      - For each p Choose the article with highest reward expectation
      - Observe the reward∈{clicked or not} for all workers
  - End the loop for p
  - Compute: $\mathrm{V_{t+1,1}} = \mathrm{V_{t,1}} + \sum_{p=1}^{P} x_{t,p,a_t}^T\, x_{t,p,a_t}$
  - Compute: $\mathrm{b_t} = \mathrm{b_{t-1}} + \sum_{p=1}^{P} x_{t,p,a_t}\, r_{t,p}$
  - Compute Click-Through-Rate: $CTR_t = \frac{\{(\mathrm{CTR_{t-1}} * (t-1) * P + (\sum_{p=1}^{P} \mathrm{r_{t,p}}))\}}{P*t}$
- End loop for time t

Experiment Results and Analysis:

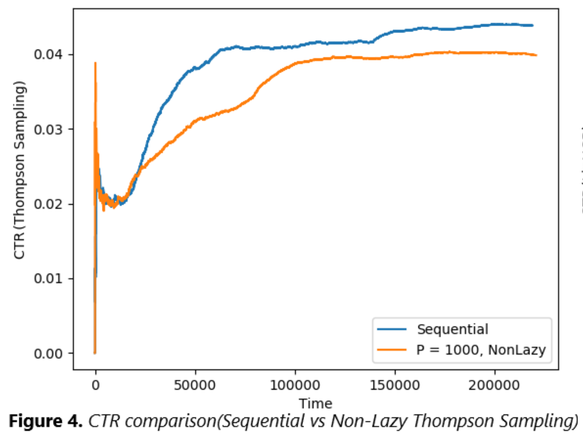

Figure 4. *CTR comparison(Sequential vs Non-Lazy Thompson Sampling)*

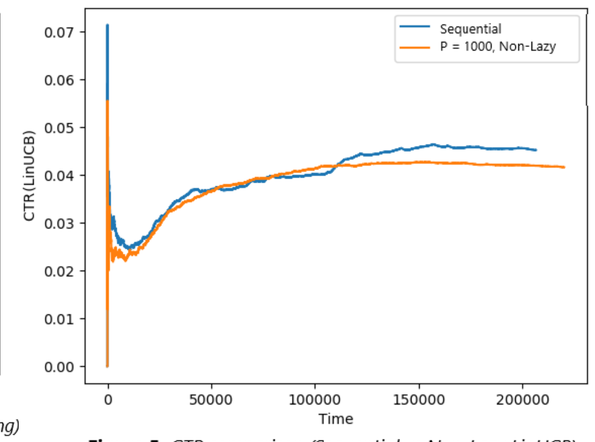

Figure 5. *CTR comparison (Sequential vs Non-Lazy LinUCB)*

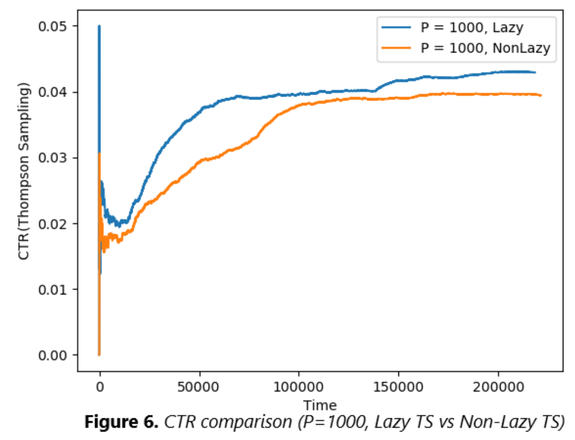

Figure 6. *CTR comparison (P=1000, Lazy TS vs Non-Lazy TS)*

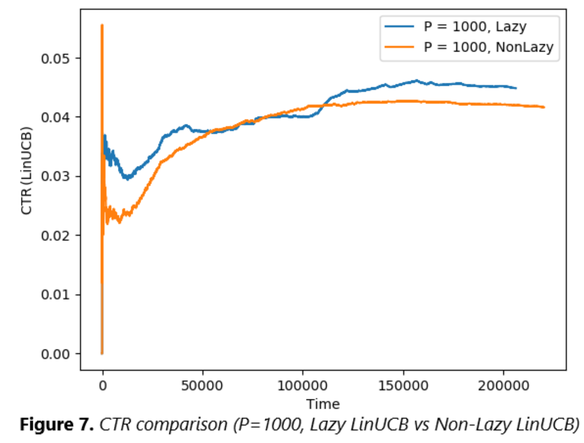

Figure 7. *CTR comparison (P=1000, Lazy LinUCB vs Non-Lazy LinUCB)*

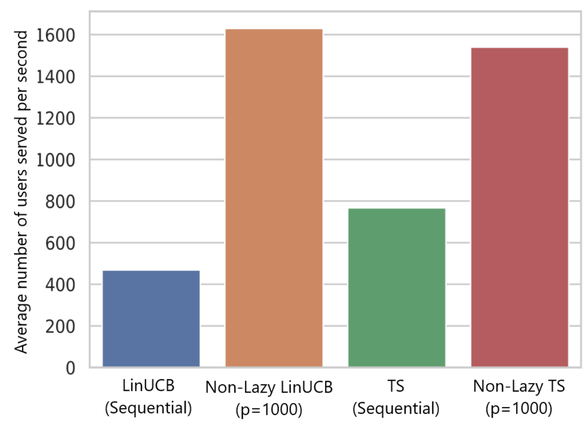

Figure 8. *Speed comparison (Sequential vs Non Lazy Algorithms)*

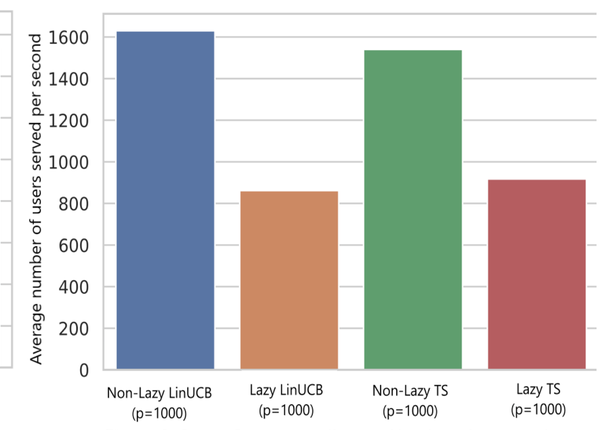

Figure 9. Speed Comparison (*Lazy vs Non-Lazy Algorithms)*

As illustrated by the plots in **Fig 4** and **Fig 5**, Sequential algorithms slightly outperform their parallel counterpart in terms of Click-Through-Rate. Considering that the 1000 workers in a batch in the Non-Lazy counterparts are unaware of the choices and rewards of the other workers in the same batch, the loss in reward is expected. But, since the parallelized algorithms in non-lazy parallel algorithms do not have to calculate covariance matrix after serving every user, as illustrated in figure 8, the algorithm can recommend articles to about **3.48 times** as many more users per second in case of LinUCB and to around **2.00 times** as many more users in case of Thompson sampling. Comparison between lazy and non-lazy versions of the parallel algorithms reveal that for both LinUCB and Thompson Sampling the CTR converges to a higher value for lazy versions of the same algorithm (illustrated in **Figure 6** and **Figure 7**). This is due to the awareness of workers in a batch of article selection of previous workers in the same batch. However, since this awareness comes from updating the covariance matrix within the batch, this addition computation leads lazy algorithm for LinUCB being around **1.89 times** faster and Lazy-Thompson Sampling being around **1.68 times** faster .

Conclusion
Parallelization evidently leads to a significant gain in speed at the expense of small loss in reward. However, the ability for a recommendation service to host a vast number of users in a short time is in t he interest of service providers, therefore, the small loss in reward is worth sacrificing when a vast number of users require to be served.