Online Parameter Selection for Web-based Ranking Problems

Deepak Agarwal
LinkedIn Corporation
dagarwal@linkedin.com

Kinjal Basu
LinkedIn Corporation
kbasu@linkedin.com

Souvik Ghosh
LinkedIn Corporation
sghosh@linkedin.com

Ying Xuan
LinkedIn Corporation
yxuan@linkedin.com

Yang Yang
LinkedIn Corporation
yyang@linkedin.com

Liang Zhang
LinkedIn Corporation
lizhang@linkedin.com

ABSTRACT

Web-based ranking problems involve ordering different kinds of items in a list or grid to be displayed in mediums like a website or a mobile app. In most cases, there are multiple objectives or metrics like clicks, viral actions, job applications, advertising revenue and others that we want to balance. Constructing a serving algorithm that achieves the desired tradeoff among multiple objectives is challenging, especially for more than two objectives. In addition, it is often not possible to estimate such a serving scheme using offline data alone for non-stationary systems with frequent online interventions.

We consider a large-scale online application where metrics for multiple objectives are continuously available and can be controlled in a desired fashion by changing certain *control* parameters in the ranking model. We assume that the desired balance of metrics is known from business considerations. Our approach models the balance criteria as a composite utility function via a Gaussian process over the space of control parameters. We show that obtaining a solution can be equated to finding the maximum of the Gaussian process, practically obtainable via Bayesian optimization. However, implementing such a scheme for large-scale applications is challenging. We provide a novel framework to do so and illustrate its efficacy in the context of LinkedIn Feed. In particular, we show the effectiveness of our method by using both offline simulations as well as promising online A/B testing results. At the time of writing this paper, the method described was fully *deployed* on the LinkedIn Feed.

CCS CONCEPTS

• **Information systems → Content ranking**; *Social networks*; • **Theory of computation → Gaussian processes**;

KEYWORDS

Bayesian Optimization; Thompson Sampling; Gaussian Processes; Online Feed Ranking

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3219847

ACM Reference Format:
Deepak Agarwal, Kinjal Basu, Souvik Ghosh, Ying Xuan, Yang Yang, and Liang Zhang. 2018. Online Parameter Selection for Web-based Ranking Problems. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3219847

1 INTRODUCTION

Webpages and mobile apps often display a set of different items in a list or grid form. In many cases, items are of different types and serve different objectives. For example, a web search typically returns organic results that are relevant to the search query and sponsored results for revenue generation. Similarly, a news feed for a member[1] in a social network might include updates from the member's network that are meant to drive direct member engagement and in some cases drive additional downstream engagement that accrues through actions like re-shares, comments, and likes. It also includes recommendations for new connections to grow the network, native ads for revenue generation, general news for engagement, job applications, online course consumption, and others. Member actions on various things have a different impact to the eco-system and there is usually a need to balance them well.

To solve problems of this kind, a Multi-Objective Optimization (MOO) framework is often used [1, 2]. The MOO framework assigns utilities for the different objectives and uses predicted values of utilities for ranking. A reasonable approach for balancing multiple objectives is to rank items using weighted linear combinations of the predicted objective utilities [1]. Finding the weights (often called tuning or control parameters) of the linear combination is challenging. Practitioners often rely on running multiple A/B experiments with various values of the control parameters and select the one that works best based on the observed values of the metrics in the experiments. This is a manual and time-consuming process which often needs to be repeated when there is a change in the environment.

This problem is exacerbated in many modern companies managing websites or apps with large traffic that are fast-evolving and dynamic. The rate of change of environment is often very high. Some examples of changes in the environment include a launch of a new item type, a launch of a new ranking model and so forth. In such situations, lengthy and repetitive manual processes add significant cost in terms of developer productivity. In fact, in many cases, such large-scale systems have dependencies on other systems that are maintained by a different organization in the company. Large-scale changes in dependent systems have a significant impact. In

[1] We shall use member and user interchangeably in this paper

these scenarios, an automatic procedure that economically adjusts the control parameter in response to change in dependencies is highly desirable.

In this paper, we introduce such a method to automate the process of finding the optimal value of the control parameters in the ranking model that meets the desired tradeoff of multiple business objectives of websites. As in the MOO framework set up in [1, 2], we assume that there is one primary utility and several other secondary utilities. With any change in the environment, the objective is to maximize the primary utility under the constraint that the secondary utilities do not drop below a pre-specified threshold. This setup can be converted to a problem of maximizing a 1-dimensional function composed of the metrics of interest over a set of parameters to tune. We apply a slight variation of the well-known Thompson Sampling (TS) approach [28] to efficiently explore the control parameter space and exploit the areas that may potentially lead to the maximum of the function. Assuming the function to maximize is smooth, we model it as a Gaussian process with its parameters updated in an online fashion as we continuously obtain data from the explore-exploit.

Our motivating application is the problem of ranking items in the LinkedIn Feed. For an overview of this problem, please see [3] and [4]. We note that our proposed method is general and can be applied to other websites with similar problems. Through online A/B testing experiments, we show that our solution performs significantly better than an approach of manually estimating the control parameters, thereby drastically reducing the effort and time required to accommodate every change in the LinkedIn environment. In addition to increasing productivity, our approach is able to find better Pareto optimal solutions.

The rest of the paper is organized as follows. We start by introducing the LinkedIn Feed in Section 2. We then formally introduce the ranking problem in Section 3. The algorithm used to solve the problem is described in Section 4. In Section 5, we talk about the engineering architecture used to deploy our solution in production. The offline and online experimental results are discussed in Section 6, followed by some of the related works in Section 7. We finally conclude with a discussion in Section 8.

2 THE LINKEDIN FEED

LinkedIn Feed is the primary place where its 500M+ members keep in touch with their connections, stay informed and advance their careers. Examples of such content types include professional updates shared by the member's connections, native ads posted by advertisers, job recommendations, professional news, new connection recommendations and courses to learn (see Figure 1). Due to the heterogeneous and dynamic nature of the content, it is essential to personalize the recommendations for each member. As each content type usually is associated with a business metric to optimize for, an efficient feed item ranking model needs to consider the tradeoff amongst the various metrics. In this paper, we consider the following three: [2]

- *Viral Actions* (VA): This metric counts the fraction of feed sessions where a user has either liked, commented or shared a feed item. This is the primary metric since it creates a ripple effect of inducing downstream actions.
- *Engaged Feed Sessions* (EFS): This metric counts the fraction of the total number of sessions where a user was active on the feed. Activity is counted as either liking, commenting or sharing on a feed item as well as clicking on anything or scrolling more than 10 feed items.
- *Job Applies* (JA): This metric measures the fraction of sessions where the user had applied for a recommended job on the feed.

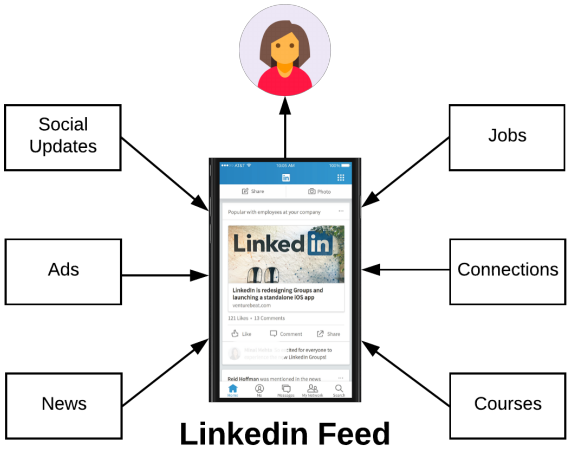


Figure 1: LinkedIn Feed: A heterogenous ecosystem

Maintaining the desired balance among these metrics is a challenging task, as the feed ecosystem is very dynamic. When a member comes to the LinkedIn Feed, the feed ranking service named `Feed-Mixer` recommends items to the member. As we have different types of items, `Feed-Mixer` is architected to use tiered ranking where the first tier calls a set of different downstream services to retrieve the top-ranked items for each type, and the second tier *re-ranks* them. For example, we have different services that provide job recommendations, news recommendations and relevant posts and shares from a member's connections. If any of the tier one ranking models change significantly, `Feed-Mixer` needs to adjust tier two to keep the desired balance of business metrics. Also, note that members can take different types of actions on an item. A member can click and read an article, apply for a job, like or comment on an update from a connection etc. While they are all indicators of engagement, the actions have different eco-system utilities. For instance, a job application helps a member find new opportunity (JA), while a comment on a news item propagates it to the member's network (VA).

Next, we discuss how `Feed-Mixer` ranks different types of items to simultaneously provide a relevant feed experience and balance different types of business metrics. For a member m, `Feed-Mixer`

[2] At the time of writing this paper, the other salient metrics were being controlled using simple rules that were quite stable (although possibly sub-optimal in terms of Pareto optimality) and did not interact much with the three metrics below. Our goal is to apply this approach to all metrics we want to automatically balance on LinkedIn Feed. This is left as future work.

computes the score of an eligible item u by

$$S(m,u) := P_{VA}(m,u) + x_{EFS}P_{EFS}(m,u) + x_{JA}P_{JA}(m,u). \quad (1)$$

Here

- $P_{EFS}(m,u)$ is the probability that m will engage with the feed item u. See [3] and [4] for details of this model.
- $P_{VA}(m,u)$ is the probability that m will take a viral action on u. This model uses similar features to the engagement model but the positive responses are only limited to likes, shares and comments.
- $P_{JA}(m,u)$ is the probability that m will apply for job u. If the item u is not a job, $P_{JA}(m,u) = 0$. See [11] for details.

The estimates of these probability measures are from independent response prediction models and available to the `Feed-Mixer` in real-time at the time of scoring. The weight vector $\boldsymbol{x} = (x_{EFS}, x_{JA})$ *controls* the balance of the business metrics EFS, VA and JA. We need a method to estimate and update this over time. For this, we consider the MOO formulation with Viral Actions as primary metric and the other two as secondary metrics described below.

$$\begin{array}{ll} \underset{\boldsymbol{x}\in\mathcal{X}}{\text{Maximize}} & VA(\boldsymbol{x}) \\ \text{subject to} & EFS(\boldsymbol{x}) \geq c_{EFS}, JA(\boldsymbol{x}) \geq c_{JA} \end{array} \quad (2)$$

where the constraints c_{EFS} and c_{JA} are business goals set by product owners. The metrics are a smooth function of control parameter $\boldsymbol{x}$ but finding the optimal $\boldsymbol{x}^*$ that solves (2) is non-trivial.

In the past, a developer spent days to accurately estimate $\boldsymbol{x}$ with desired behavior with every significant drift and intervention. Using the approach described in this paper, we have completely eliminated the laborious manual tuning, significantly increased developer productivity as well as obtained better Pareto optimal solutions.

3 PROBLEM DEFINITION

In this section, we develop our approach in a generic setting and then use LinkedIn Feed as a specific example to illustrate the idea.

Suppose $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d$ is a d-dimensional control parameter vector that controls the composition of items on a webpage, and $\mathcal{X}$ is a compact subset of $\mathbb{R}^d$. Let the multiple utilities associated with the webpage be:

$$U_i(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}_+ \text{ for } i = 1, \ldots, n.$$

Every time the webpage is displayed for a given $\boldsymbol{x} \in \mathcal{X}$, we can collect data that help us estimate the utilities at $\boldsymbol{x}$. Examples of such utilities for LinkedIn Feed include VA, EFS, JA and so forth. We assume that each $U_i(\boldsymbol{x})$ is smooth and continuous in $\boldsymbol{x} \in \mathcal{X}$.

Without loss of generality, we assume that there is a primary utility, $U_1(\boldsymbol{x})$, and other secondary utilities. This means that our objective is to maximize $U_1(\boldsymbol{x})$ under the constraint that the other utilities are above a pre-specified threshold, i.e., we wish to solve the following optimization problem:

$$\begin{array}{ll} \underset{\boldsymbol{x}\in\mathcal{X}}{\text{Maximize}} & U_1(\boldsymbol{x}) \\ \text{subject to} & U_i(\boldsymbol{x}) \geq c_i \qquad \text{for } i = 2, \ldots, n. \end{array} \quad (3)$$

We can rewrite the MOO problem defined in (3) in a simpler form by introducing the Lagrangian weights:

$$\underset{\boldsymbol{x}\in\mathcal{X}}{\text{Maximize}}\ U_1(\boldsymbol{x}) + \sum_{i=2}^{n} \lambda_i \sigma_\xi(U_i(\boldsymbol{x}) - c_i) \quad (4)$$

where $\sigma_\xi(z) = (1 + \exp(-\xi z))^{-1}$ for some large ξ denotes the smoothing function which take the value close to 1 for positive z and close to 0 for negative z. Thus for the i-th satisfied constraint we add λ_i to the function value and if i is not satisfied, we add 0. The trick is to let the value of λ_i help us identify how many constraints are satisfied or violated. Specifically, since each U_i is continuous and $\mathcal{X}$ is compact, we have

$$\sup_{\boldsymbol{x}\in\mathcal{X}} U_i(\boldsymbol{x}) \leq B \qquad \text{for } i = 1, \ldots, n,$$

for some $B > 0$. Thus, we can set $\lambda_i > B$ (for example $\lambda_i = \exp(B)$) to help us identify if we have reached the maximum while satisfying all the constraints or not. The maximum after satisfying all constraints should lie in the range $((n-1)\exp(B), B + (n-1)\exp(B))$. If we violate $j \in \{0, \ldots, n-1\}$ constraints, then the optimized value lies between $((n-1-j)\exp(B), B + (n-1-j)\exp(B))$. This helps us to clearly identify if we have been able to achieve a feasible solution or not. This neat trick also allows us to reformulate problem (3) as

$$\underset{\boldsymbol{x}\in\mathcal{X}}{\text{Maximize}}\ U_1(\boldsymbol{x}) + \exp(B) \sum_{i=2}^{n} \sigma_\xi(U_i(\boldsymbol{x}) - c_i), \quad (5)$$

which brings us to the familiar form of a global optimization problem of an unknown function, in which case we are trying to solve

$$\underset{\boldsymbol{x}\in\mathcal{X}}{\text{Maximize}}\ f(\boldsymbol{x}), \quad (6)$$

where

$$f(\boldsymbol{x}) := U_1(\boldsymbol{x}) + \exp(B) \sum_{i=2}^{n} \sigma_\xi(U_i(\boldsymbol{x}) - c_i)$$

is a smooth and continuous function in $\boldsymbol{x}$. We assume there is an unique $\boldsymbol{x}^*$ that maximizes f:

$$\boldsymbol{x}^* := \underset{\boldsymbol{x}\in\mathcal{X}}{\text{argmax}} f(\boldsymbol{x}). \quad (7)$$

3.1 LinkedIn Feed Use Case

For LinkedIn Feed, the parameters to learn are $\boldsymbol{x} = (x_{EFS}, x_{JA})$ in a reasonable pre-determined range $\mathcal{X}$. Note that we do not observe the utilities (or metrics) directly but can estimate them using the data we collect. Let $Y^k_{i,j}(\boldsymbol{x}) \in \{0, 1\}$ denotes whether member i has an action $k = VA, EFS$ or JA in the j-th session when her feed was served with parameter $\boldsymbol{x}$. We model each of these random variables as

$$Y^k_{i,j}(\boldsymbol{x}) \sim Bernoulli(\sigma(f_k(\boldsymbol{x})))$$

where $\sigma(\cdot)$ is the logit function and f_k is assumed to come from a Gaussian Process prior with mean 0 and covariance kernel K^k_η for $k = VA, EFS, JA$. We also assume independence over sessions j. Thus, for each $k = VA, EFS, JA$,

$$Y^k_i(\boldsymbol{x}) \sim Binomial(n_i(\boldsymbol{x}), \sigma(f_k(\boldsymbol{x}))),$$

where $n_i(\boldsymbol{x})$ is the number of sessions during the experiment for member i with $\boldsymbol{x}$. Thus, for each parameter $\boldsymbol{x}$, we observe the

aggregated data $Y_i^k(\boldsymbol{x})$ and $n_i(\boldsymbol{x})$ for $k = VA, EFS, JA$. We can estimate the utilities as

$$\begin{aligned} VA(\boldsymbol{x}) &= \sigma(f_{VA}(\boldsymbol{x})) \\ EFS(\boldsymbol{x}) &= \sigma(f_{EFS}(\boldsymbol{x})) \\ JA(\boldsymbol{x}) &= \sigma(f_{JA}(\boldsymbol{x})) \end{aligned} \tag{8}$$

We note that the utilities $VA(\boldsymbol{x})$, $EFS(\boldsymbol{x})$ and $JA(\boldsymbol{x})$ are bounded by 1 for each member session, hence, λ_i being any value greater than 1 would be sufficient to provide the property of identifying how many constraints are satisfied. Therefore, equation (4) can be simplified to the following:

$$\underset{x \in \mathcal{X}}{\text{Maximize}}\ VA(\boldsymbol{x}) + \lambda(\sigma_{\xi}(EFS(\boldsymbol{x}) - c_{EFS}) + \sigma_{\xi}(JA(\boldsymbol{x}) - c_{JA})), \tag{9}$$

where any constant value of $\lambda > 1$ would suffice.

4 CONTINUUM-ARMED THOMPSON SAMPLING

In this section, we detail out the method of solving the generic problem (7) through a modified version of the Thompson Sampling algorithm [28]. The method described here will then be used to solve problem (9) in the subsequent sections. The main idea is similar to an explore-exploit algorithm, where at every iteration we identify a set to points at which we evaluate the function f and based on the obtained data, we try to identify the next best set of points. The algorithm is designed to converge to the global maximizer of f.

4.1 Gaussion Processes and Kernel Functions

To solve the global optimization problem (7), we need sufficient smoothness assumptions on f. This is enforced by modeling f as a sample from a Gaussian Process (GP) with mean 0 and kernel $k(\cdot, \cdot)$. For any $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, let $\boldsymbol{f}$ denote the vectorized version of the function values obtained at the n points. That is,

$$\boldsymbol{f} = (f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n))^T.$$

Then, $\boldsymbol{f}$ is multivariate normal with mean 0 and covariance K, where $K_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. We assume that k is a Mercer kernel on the space $\mathcal{X}$ with respect to the uniform measure on $\mathcal{X}$.

A common example of a kernel is the Gaussian RBF-kernel, which can be parametrized by η, i.e.

$$k_{\eta}(\boldsymbol{x}, \boldsymbol{x}') = \zeta^2 \exp\left(-\frac{1}{2}\sum_{i=1}^{d}\frac{(x_i - x_i')^2}{\ell_i^2}\right),$$

with $\eta = (\zeta, \ell_1, \ldots, \ell_d)$. See [21] for more details on Mercer's Theorem, kernel smoothing and many other examples.

4.2 Thompson Sampling

Suppose D_t denotes the data we have till stage $t-1$ and $\mathcal{F}_t$ denotes the posterior of the maximizer of f given D_t. The Thompson Sampling approach samples a new data point $\boldsymbol{x}_t$ at iteration t from $\mathcal{F}_t$. We observe data $y_t \sim \mathcal{P}_{\theta}(f(\boldsymbol{x}_t))$, where $\mathcal{P}_{\theta}$ is a distribution of observed data parametrized by θ. Once we observe the data, we update $D_{t+1} = D_t \cup \{(\boldsymbol{x}_t, y_t)\}$. We initialize the process by assuming a non-informative prior on the distribution of the maximizer, i.e., $\mathcal{F}_0 = U(\mathcal{X})$, the uniform distribution on $\mathcal{X}$. We stop the procedure when the variance of the distribution of $\mathcal{F}_t$ becomes considerably small and we return $mode(\mathcal{F}_t)$ as the estimate of the global maximizer of f.

In some cases, especially when the variance of the observed data is large, this process might converge to a local optimum. Since we sample $\boldsymbol{x}_t$ from $\mathcal{F}_t$, we might get stuck in one place and not explore the entire space. To ensure the convergence to the global maximum, we consider an ϵ-greedy approach. That is, at every iteration t with some probability $\epsilon > 0$ (usually small), we explore the entire region $\mathcal{X}$ uniformly. Thus, we sample $\boldsymbol{x}_t \sim \mathcal{F}_t$ with probability $1 - \epsilon$ and we sample $\boldsymbol{x}_t \sim U(\mathcal{X})$ with probability ϵ. The algorithm also depends on estimating the kernel and distributional hyperparameters (described in Section 4.3) as well as on the posterior distribution of f given the data D_t (described in Section 4.4). We state the detailed steps in Algorithm 1.

Algorithm 1 Thompson Sampling for Infinite-Armed Bandits

1: Input : Function f, Kernel form k_{η}, Domain $\mathcal{X}$, Parameter γ, threshold ϵ
2: Output : $\hat{\boldsymbol{x}}^*$, the global maximum of f
3: Sample $\boldsymbol{x}_0$ uniformly from $\mathcal{X}$
4: Observe y_0
5: Put $D_1 = \{(\boldsymbol{x}_0, y_0)\}$
6: **for** $t = 1, 2, \ldots$ **do**
7: Estimate the kernel and distribution hyperparameters η_t^*, θ_t^* using D_t as detailed in Section 4.3
8: Given these optimal hyperparameters, estimate the posterior mode $\hat{\boldsymbol{f}}_t$ according to (12)
9: Using D_t and $\hat{\boldsymbol{f}}_t$ generate the posterior distribution as detailed in Section 4.4
10: Pick a random number $\rho \in [0, 1]$
11: Draw sample f_t from the posterior distribution
12: **if** $\rho > \epsilon$ **then**
13: Set $\boldsymbol{x}_t = \text{argmax}_{\boldsymbol{x} \in \mathcal{X}} f_t(\boldsymbol{x})$
14: **else**
15: Sample $\boldsymbol{x}_t \sim U(\mathcal{X})$
16: **end if**
17: Observe y_t at $\boldsymbol{x}_t$
18: Set $D_{t+1} = D_t \cup \{(\boldsymbol{x}_t, y_t)\}$
19: Break from the loop when $var(\mathcal{F}_t) < \gamma$.
20: **end for**
21: **return** $\hat{\boldsymbol{x}}^* := mode(\mathcal{F}_t)$

Algorithm 1 aims to find the true maximum of the sampled function on Line 13. However, in practice, we can only find this over a grid of values. We choose a large set of points that span the entire space $\mathcal{X}$ and find the maximum over these set of points. Specifically, we use the Sobol points [14] to span this space. These are a set of low-discrepancy points which are very easy to construct in any dimension and are equidistributed on the unit square.

Moreover, since we are only working on a grid, to find the true global maximum we follow a recursive zooming approach. At every stage, we have a fixed search grid. Once we have found that the algorithm has converged to a small enough region, we reduce the search space and generate more points within it and keep searching. We keep the old points in this region as supporting prior data. We stop the algorithm when the objective value does not change further

through this zooming mechanism. This addition to Algorithm 1 is currently deployed in production and serving live traffic.

Remark. We state Algorithm 1 to find the maximizer of a Gaussian process. Note that the same algorithm will work for any smooth transformation of the Gaussian processes as well. We use this fact in our application in later sections.

Remark. It is known that Algorithm 1 converges to the true $\boldsymbol{x}^*$. We refer the reader to [9] for details of the theoretical considerations of Algorithm 1. In Sections 4.3 and 4.4, we describe the estimation of hyperparameters and posterior distribution used in Algorithm 1. Their derivation follows from [22]. Readers can skip these on first reading and move to Section 5 without loss of continuity.

4.3 Estimation of Hyperparameters

There are several methods known in literature for estimating the hyperparameters in this set up. We focus on the maximum a posteriori (MAP) estimation. For other methods see [29]. Here we use,

$$\begin{aligned}\{\hat{\eta}_t, \hat{\theta}_t\} &= \underset{\eta,\theta}{\operatorname{argmax}}\, p(\eta, \theta | D_t) \\ &= \underset{\eta,\theta}{\operatorname{argmin}} \left[-\log p(D_t|\eta,\theta) - \log p(\eta,\theta)\right], \end{aligned} \tag{10}$$

where $p(\cdot)$ denotes the likelihood function. For the Gaussian RBF kernel and if $\mathcal{P}_\theta(\mu) = N(\mu, \theta^2)$, we can write the marginal likelihood given the parameters, $p(D_t|\eta,\theta) = \int p(\boldsymbol{y}|\boldsymbol{f},\theta)p(\boldsymbol{f}|\boldsymbol{x},\eta)\mathrm{d}\boldsymbol{f}$ in a closed form,

$$\log p(D_t|\eta,\theta) = C - \frac{1}{2}\log\left|K_\eta + \theta^2 I\right| - \frac{1}{2}\boldsymbol{y}^T\left(K_\eta + \theta^2 I\right)^{-1}\boldsymbol{y},$$

where $\boldsymbol{y}$ denotes the vectorized version of our observed function values. Since this function is easily differentiable in θ and η, we can find the optimum using any gradient descent algorithm [12].

However, in many practical situations, $\mathcal{P}_\theta$ is not the Gaussian distribution. In such situations, a closed form expression cannot be found and hence, we resort to Laplace Approximations or EP's marginal likelihood approximation [29]. For example, if we observe $y_t \sim \mathcal{P}_\theta(f(\boldsymbol{x}_t))$, for some non-Gaussian distribution $\mathcal{P}_\theta$, then we work with the quasi-likelihood instead of the likelihood.

$$\begin{aligned}\log p(D_t|\eta,\theta) &\approx \log q(\boldsymbol{y}|X,\eta,\theta) \\ &= -\frac{1}{2}\hat{\boldsymbol{f}}^T K_\eta^{-1}\hat{\boldsymbol{f}} + \log p_\theta(\boldsymbol{y}|\hat{\boldsymbol{f}}) - \frac{1}{2}\log|B|, \end{aligned} \tag{11}$$

where $B = I + W^{1/2}K_\eta W^{1/2}$, $\hat{\boldsymbol{f}}$ is the maximum of the posterior equation:

$$\hat{\boldsymbol{f}} = \underset{f}{\operatorname{argmax}} \quad \log p_\theta(\boldsymbol{y}|\boldsymbol{f}) - \frac{1}{2}\boldsymbol{f}^T K_\eta^{-1}\boldsymbol{f} - \frac{1}{2}\log|K_\eta| \tag{12}$$

and W is the diagonal matrix $W = -\nabla\nabla\log p_\theta(\boldsymbol{y}|\hat{\boldsymbol{f}})$. Thus to identify the optimal kernel hyperparameter in these general situations, we follow Algorithm 2. For more details and the derivation see [22].

4.4 Posterior Distribution

Given (η_t^*, θ_t^*), the optimal kernel and distributional hyperparameters, we estimate the posterior mode $\hat{\boldsymbol{f}}$ using (12). In addition, we

Algorithm 2 Estimating Kernel and Distributional Hyperparameters

1: Input : Data D_t upto iteration $t-1$
2: Output : (η_t^*, θ_t^*), the optimal kernel and distribution hyperparameters for iteration t.
3: Choose initial hyperparameter η_0, θ_0.
4: **for** $i = 0, 1, 2, \ldots$ **do**
5: Evaluate kernel K_{η_i}.
6: Generate $\hat{\boldsymbol{f}}_i$ using K_{η_i} and θ_i by Algorithm 3.1 from [22]
7: Using $\hat{\boldsymbol{f}}_i$ and gradient descent on the quasi-likelihood (11), generate the next choice of hyperparameters η_{i+1}, θ_{i+1}.
8: Break from the loop when η_i, θ_i converges to some value η_t^*, θ_t^*.
9: **end for**
10: **return** (η_t^*, θ_t^*)

can now estimate the mean and covariance function of the posterior of f given the data D_t. Specifically, the mean function defined as

$$\mu_t(\boldsymbol{x}) = \begin{cases} k_{\eta_t^*}(\boldsymbol{x})^T K_{\eta_t^*}^{-1}\boldsymbol{y} & \text{if } \mathcal{P}_\theta(\cdot) \sim N(\cdot, \theta^2) \\ k_{\eta_t^*}(\boldsymbol{x})^T K_{\eta_t^*}^{-1}\hat{\boldsymbol{f}} & \text{otherwise} \end{cases} \tag{13}$$

and the covariance kernel and variance is defined as

$$\begin{aligned} k_t(\boldsymbol{x},\boldsymbol{x}') &= k_{\eta_t^*}(\boldsymbol{x},\boldsymbol{x}') - k_{\eta_t^*}(\boldsymbol{x})^T(K_{\eta_t^*} + W^{-1})^{-1}k_{\eta_t^*}(\boldsymbol{x}') \\ \sigma_t^2(\boldsymbol{x}) &= k_t(\boldsymbol{x},\boldsymbol{x}) \end{aligned} \tag{14}$$

where

$$k_{\eta_t^*}(\boldsymbol{x}) = (k_{\eta_t^*}(\boldsymbol{x},\boldsymbol{x}_1), \ldots, k_{\eta_t^*}(\boldsymbol{x},\boldsymbol{x}_n))^T$$

and

$$W = \begin{cases} \frac{1}{\theta_t^{*2}}\boldsymbol{I} & \text{if } \mathcal{P}_\theta(\cdot) \sim N(\cdot, \theta^2) \\ -\nabla\nabla\log p_{\theta_t^*}(\boldsymbol{y}|\hat{\boldsymbol{f}}) & \text{otherwise} \end{cases}$$

The posterior of f given the data D_t follows a Gaussian Process distribution with a mean $\mu_t(\boldsymbol{x})$, covariance $k_t(\boldsymbol{x},\boldsymbol{x}')$ and variance $\sigma_t^2(\boldsymbol{x})$. For more details and their derivation, we refer to [22].

5 SYSTEM ARCHITECTURE

In this section, we describe the system architecture for automatically tuning parameters to optimize (9) for the LinkedIn Feed. We start with a detailed description of how our method gets implemented in Section 5.1 and then discuss several practical design considerations in Section 5.2.

5.1 Implementation of Thompson Sampling Algorithm

We illustrate the schematic design of our system in Figure 2, which also applies to other web-based recommendation systems. It contains two main components: an offline component responsible for data processing and model parameter updates, and an online component that lives in the `Feed-Mixer` service and is responsible for model serving.

5.1.1 Offline Component. The offline component runs every hour and consists of the following steps:

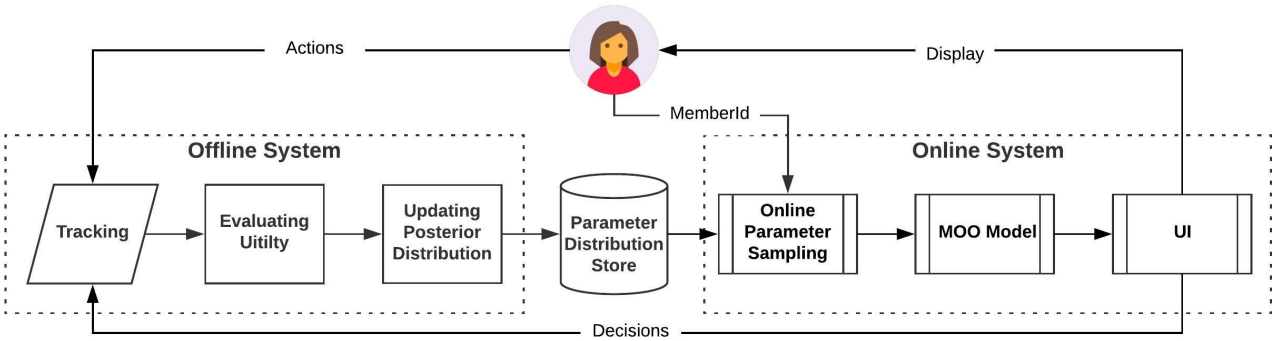


Figure 2: System Architecture

- `Tracking`: We track various things when a member interacts with the LinkedIn webpage or the mobile app – we track the items we serve to the member, what control parameters were used to rank the items served and the member actions in the session (click, like, apply for job, etc.) using `Kafka` events. See [27] for details on `Kafka`.
- `Evaluating Utility`: The metrics (or utilities) of interest (EFS, Viral Actions, and Job Applies) are calculated periodically based on the tracking data. We also aggregate the metric values per parameter value $\boldsymbol{x}$ used in the last time period.
- `Updating Posterior Distribution`: We use the metric observations to update the posterior distribution of the maximizer of (9) to obtain $\mathcal{F}_t$. We repeat steps 11 - 16 in Algorithm 1 multiple times to generate samples from $\mathcal{F}_t$. We discretize the domain $\mathcal{X}$ and express the posterior distribution $\mathcal{F}_t$ as a discrete distribution. This distribution is stored in a Voldemort-based key-value store [27] that powers our online services.

5.1.2 *Online Component in* `Feed-Mixer`.

- `Online Parameter Sampling`: When a member visits the LinkedIn Feed, this module returns a sample from the empirical distribution $\mathcal{F}_t$. Assume, we have N points $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ with sampling probability $(p_1, \ldots, p_N)$ where $\sum_{i=1}^{N} p_i = 1$ that forms the empirical distribution $\mathcal{F}_t$. As there is no inherent ordering in the $\boldsymbol{x}_i$'s, we pick any order and create the empirical CDF, $(c_1, \ldots, c_N)$ where $c_i - c_{i-1} = p_i$. Since every member at LinkedIn can be identified by a unique `memberId`, we map it to $z \in [0, 1]$ using MD5[23]. If $c_{i-1} < z \leq c_i$, we set $\boldsymbol{x}_i$ as the sampled parameter for the member.
- `MOO Models`: The parameter value is then used in the `MOO models` to rank items that `Feed-Mixer` receives from other services using (1). The ranked list is sent to the UI to be rendered to the member on the website or the mobile app.

5.2 Practical Design Considerations

We discuss some of the design decisions we made to make the system work at LinkedIn's scale.

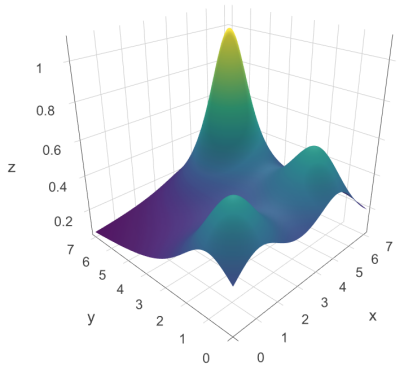
5.2.1 Online sampling latency. The parameter distribution is a cumulative distribution function, derived from thousands of the points pre-sampled offline from the true posterior distribution $\mathcal{F}_t$. Sampling from this is done by a pseudo-random generator and MD5 hashes. In detail, we use a random generator, with the `version` of the parameter distribution as the `seed`, to generate the `salt` for MD5. Every member `id` is hashed and converted to a float value in $[0, 1]$. This is much more lightweight compared to sampling from the original posterior, especially during online serving. This also creates a non-parametric mechanism for sharing the distribution from offline to online.

5.2.2 Consistency in user experience. The recommended list of items can change dramatically with updates of the parameter values. It will be a sub-optimal experience if members see very different feeds with every page refresh or sessions that are close in time. In the aforementioned sampling method, every member will get the same parameter value for the same parameter distribution.

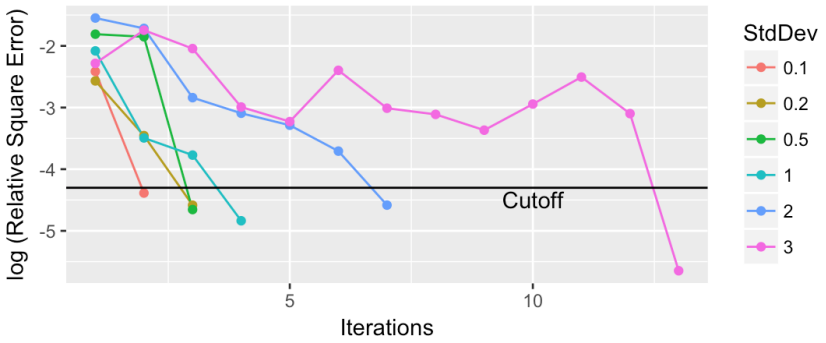
5.2.3 Offline flow frequency. Even though we describe Algorithm 1 in a manner where we update the posterior with every newly collected sample, a batch computation mode is more practical in web-scale. We set the offline flow to run at an hourly frequency. The sampling distribution is thus updated per hour, and within each hour members will expect a relatively consistent experience (except for the short period of time when the distribution is being updated).

There is another factor to consider when deciding the update frequency. If we collect data for a long period of time before updating the posterior then we will have more data and get better estimates of the quantities of interest. However, we may end up in a suboptimal sampling distribution for a long time which will hurt business metrics.

5.2.4 Independence assumption. When estimating the utilities we assume a parametric model in which we assume that the random



(a) Trimodal Shekel Function



(b) Decay of log relative square error

Figure 3: Offline simulation results of convergence on a trimodal Shekel function.

variables $Y_{ij}^{VA}(\boldsymbol{x})$, $Y_{ij}^{EFS}(\boldsymbol{x})$ and $Y_{ij}^{JA}(\boldsymbol{x})$ are independent. This is not a theoretically valid assumption, but one which works well for our particular utilities. Modeling the covariance between the utilities might result in better variance estimates of the utilities which in turn might result in faster convergence of our algorithm. However, the added complexity did not result in a good tradeoff in our application. The convergence rates we saw in practice were good enough for us since we could collect a good amount of data in each iteration. In applications where the data is scarce, modeling the covariance between utilities can be helpful.

5.2.5 Choice of business constraint thresholds. For the LinkedIn Feed, the business constraints in (2) are typically taken as the previous status quo. When something changes in the eco-system, we would like to have the secondary metrics meet the levels before the change. In practice, we also sometimes allow a small percentage drop (e.g. 1%) from the previous stage to incorporate some flexibility for our algorithm to work.

6 EXPERIMENTS

In this section, we describe in details the experiments that we have successfully run on LinkedIn Feed for our approach described in Section 4 and Section 5. We start with some offline simulation experiments in Section 6.1, as a validation of our approach. Then we show the online A/B test performance of our method in Section 6.2.

6.1 Offline Simulation Results

We validate the effectiveness of our proposed method through a simulation study, with a simple utility function in a 2-dimensional parameter space to maximize for. In particular, we pick a trimodal Shekel function [24],

$$f(\boldsymbol{x}) = \sum_{i=1}^{3}\left(c_i + \sum_{j=1}^{2}(x_j - a_{ij})^2\right)^{-1}$$

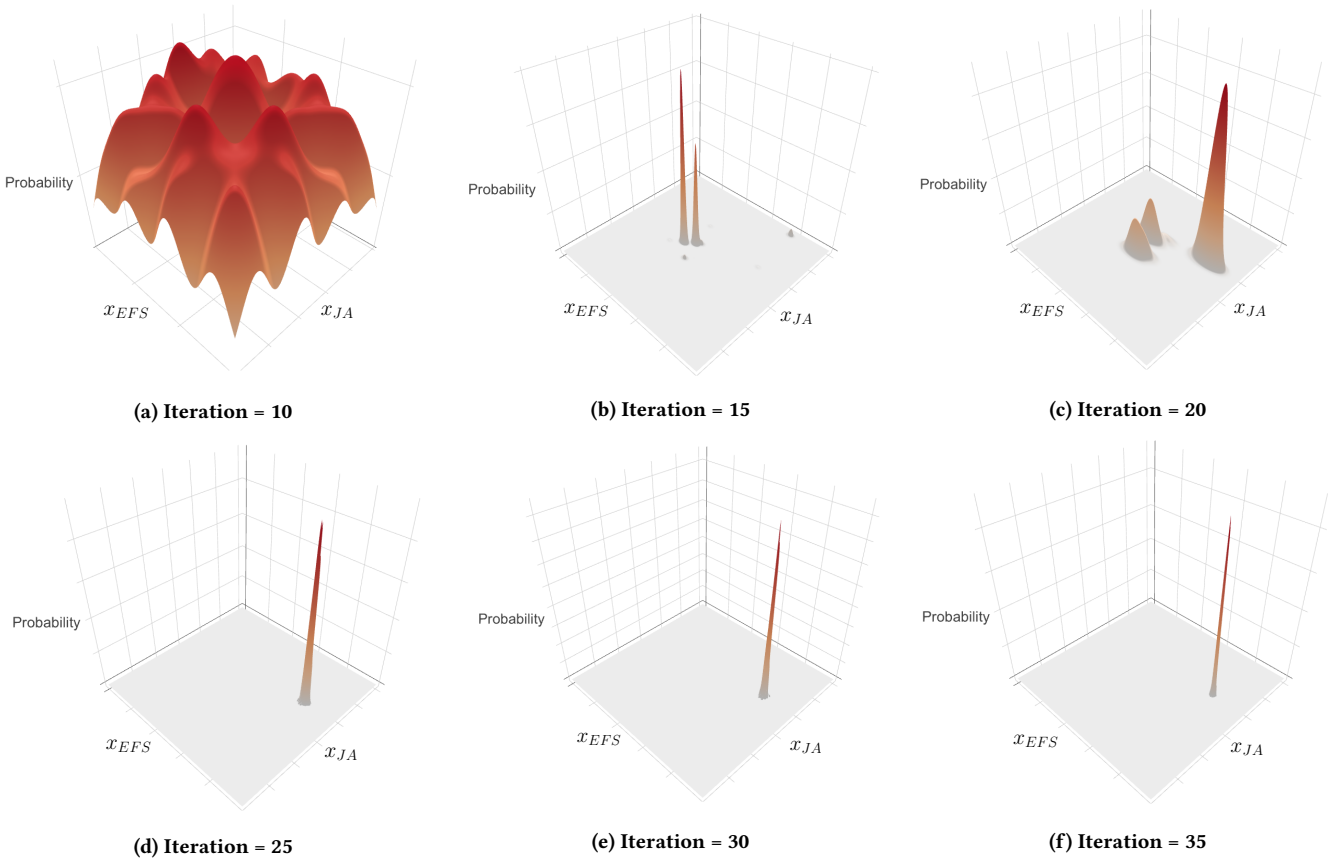
where the local maximum occurs at $\boldsymbol{a}_1 = (1,1)$ and $\boldsymbol{a}_2 = (1,5)$ and the global maximum occurs at $\boldsymbol{a}_3 = (5,5)$ (see figure 3a). We then add a Gaussian random noise with standard deviation σ to the function value to form the observations, i.e. $y(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$. Figure 3b shows the decays in the relative squared error for each of the different values of $\sigma \in [0.1, 3]$ at each iteration t, with the error defined as $2\log_{10}(\|\boldsymbol{x}_t - \boldsymbol{x}^*\| / \|\boldsymbol{x}^*\|)$. When σ is small, we observe very quick convergence. As the value of σ increases, we observe that the algorithm takes a longer time to converge but the decay rate of the relative squared error over iterations is still exponential.

6.2 Online A/B Experiments

6.2.1 Detailed Setup. Our goal is to automatically find the best $\boldsymbol{x}$ to maximize (9), with utilities VA, EFS, JA modeled as in equation (8). Since each of the utilities are bounded by 1, we choose $\lambda = 5$ in our experiments to identify if each constraint has been met. Moreover, the smoothing parameter ξ is chosen as 100, so that can have a sharp transition in the sigmoid function from 0 to 1.

To demonstrate the effectiveness of our approach, we set up two control online A/B test buckets (with different underlying prediction models), and for each bucket, the values of $\boldsymbol{x}$ were manually tuned for more than 7 days and stabilized. We denote these two control parameter settings as $\boldsymbol{x}_{c_1}$ and $\boldsymbol{x}_{c_2}$, respectively. For each control bucket, our objective in A/B test is to demonstrate that our approach can find the optimal $\boldsymbol{x}$ which maximizes VA with EFS and JA greater than threshold c_{EFS} and c_{JA}, respectively. As explained in Section 5.2, we incorporated all the practical design considerations, e.g. the bounds c_{EFS}, c_{JA} are chosen so as to allow for a 1% expected drop in those metrics compared to the control model.

6.2.2 Convergence of the Posterior Distribution. Figure 4 shows how the posterior distribution of the maximum converges as we continue the iterations in our first experiment (with control being $\boldsymbol{x}_{c_1}$). The observations from the second experiment were very similar and hence we skip them for brevity. Figure 4 highlights a lot of key attributes of Algorithm 1. We initialize the algorithm by performing quasi-random sampling for the first 10 iterations. This is done to collect the initial data. Instead of doing simple random sampling, we sample over an equidistributed set of points such that there are no large gaps in the domain. Most literature shows that



(a) Iteration = 10 **(b) Iteration = 15** **(c) Iteration = 20**

(d) Iteration = 25 **(e) Iteration = 30** **(f) Iteration = 35**

Figure 4: The convergence of the posterior distribution of the maximum as the iterations increase.

doing such sampling actually improves the accuracy of function estimation [17].

Figure 4a shows the flat prior with minor undulations. After collecting enough data we start to follow Algorithm 1. We see that within just 5 more iterations (see figure 4b), we are able to converge to a distribution which is very close to the Dirac delta, having a small variance. However, stopping here would be a mistake since, after just one more random sampling step, we see that the distribution starts to shift. Figure 4c shows this transition as the initial two peaks start to die down and we start to see a higher peak at a new location. Continuing with the iterations, this peak starts to strengthen even more. Figure 4d shows that the initial two peaks have disappeared and we just get a single peak. We continue our algorithm for many more iterations which include random samples as well. However, by iteration 30, the distribution has become very stable and the peak of the distribution remains at the same location. Figure 4e shows this effect. We continue our algorithm for some more time to ensure convergence and we see that the distribution remains very close to the same Dirac delta distribution, which is seen in Figure 4f. This experiment shows the importance of choosing the ϵ-greedy approach; without this adjustment to Thompson Sampling, there is a high probability of getting stuck at a local maximum instead of converging to the true global maximum. For more theoretical details on the effect of ϵ to the convergence of the algorithm, we refer to [9].

6.2.3 Online Metric Results. Not only did we obtain the non-trivial convergence to the global maximum, we achieved substantial lifts on the actual metrics of interest. The results of the two experiments are shown in Table 1.

Table 1: Online A/B results for Online Parameter Selection in LinkedIn Feed Ranking

Metric	Lift (%) vs Control $\boldsymbol{x}_{c_1}$	Lift (%) vs Control $\boldsymbol{x}_{c_2}$
Viral Actions	+3.3%	+1.2%
Engaged Feed Sessions	-0.8%	0%
Job Applies	+12.8%	+6.4%

The numbers in the two columns of Table 1 are all statistically significant ($p < 0.05$) and are obtained by comparing with the baseline model, which was running with fixed parameters $\boldsymbol{x}_{c_1}$ and $\boldsymbol{x}_{c_2}$ respectively. In the first experiment, we achieved 3.3% lift on Viral Actions, the most important metric for LinkedIn Feed, which has far-reaching consequences from more subsequent sessions to more revenue. We sacrificed Engaged Feed Sessions by 0.8% while

our formulation allowed for a 1% drop. On the other hand, we have seen a 12.8% lift on Job Applies in which we also allowed for a 1% drop. Moreover, it was able to converge on the optimal value within about 36 hours compared to a suboptimal $\boldsymbol{x}_{c_1}$ that took over 7 days to identify. Similarly, in next experiment, we again see a lift in our main metric of interest as well as in Job Applies. The Engaged Feed Sessions metric in this experiment remained stable and not statistically different from the different baseline even though we still allowed for a 1% drop. We were able to improve Job Applies in both experiments by converging to the true optimal which was not possible in the manual search.

6.3 Overall Learnings

Before finalizing the solution which we described above, we iterated on various strategies and we summarize some insights from our learnings below.

6.3.1 Gaussian Process Optimization. Adding priors to the hyperparameters in the model was very important. Specifically, we added an inverse-gamma prior to the kernel hyperparameter η in the learning process and this prior was updated and passed along through the iterations. Moreover, modeling the observations of different utilities as separate Gaussian processes vastly outperformed the initial choice of modeling the entire objective function in (9) as a single Gaussian process. We repeated steps 7-9 of Algorithm 1 to generate the posterior distribution for each of the three Gaussian processes. We also repeated Step 11 to sample from these three different posteriors and then combined them to form the overall utility function in (9). Lastly, by using $\hat{\boldsymbol{f}}_t$ as the starting prior to obtaining $\hat{\boldsymbol{f}}_{t+1}$, we achieved much faster convergence.

6.3.2 Choice of Objective Functions. There was once a business need to maximize EFS while controlling Viral Action and Job Applies. We observed very large variance in the distribution of EFS which made convergence very slow. However, when we changed the formulation to (9) we saw very quick convergence. We learned that the variation in the main utility has a strong impact on the convergence time. This is seen empirically in Figure 3b and also theoretically proved in [9].

7 RELATED WORK

The feed ranking problem [3, 4] has been traditionally posed as a MOO problem with known objective functions [1, 2]. Contrary to that, in this paper, we explored the approach based on global black-box optimization [18] to solve some of the pain points. Finding the global maximum of a smooth function is a well-studied problem. Most of the algorithms are based on Bayesian Optimization techniques, which work under the premise that function evaluations are expensive and noisy. Explore-exploit algorithms, such as the Gaussian Process Upper Confidence Bound (GP-UCB) [26] and many of its variants [8, 16, 19], are one class of algorithms used for optimizing unknown functions. Almost all such algorithms optimize an *acquisition function* to find the next point where we evaluate the function. Well-known acquisition functions include the Probability of Improvement (PI), Expected Improvements (EI) and Upper Confidence Bounds (UCB) [25].

However, evaluating at a single point at every iteration was not optimal in our setting, since we wanted quicker exploration. Hence, we focused on the Thompson Sampling approach. Although this is an old idea dating back to 1933, [28], there has been considerable attention in the recent past [10, 15, 20]. Several studies have shown good empirical evidence of Thompson Sampling [13] and more recently, theoretical proofs have been obtained for the multi-arm bandit setting and some generalizations [5–7]. The ϵ-greedy variant of the Thompson Sampling approach that we have introduced here also has good theoretic properties [9].

8 CONCLUSION

In this paper, we have developed a method of tuning the parameters for web-based ranking problems. We approach the problem as a global black-box optimization problem of a smooth function, where each choice of the parameter creates a potentially different ranking. We use the novel technique of ϵ-greedy Thompson Sampling to solve the problem. This method works well when we are trying to balance multiple objectives with corresponding unknown weights. We have successfully implemented the algorithm in practice and it is deployed in the engineering stack for LinkedIn Feed. We achieved substantial metric gains from this technique and drastically improved developer productivity. Moreover, the method is generic enough to be applied to any MOO problem, where we are trying to balance multiple metrics. We modeled the objective function as a smooth Gaussian process, with each metric having a latent function whose distributions are considered independent. We leave it as a future work to generalize this into a dependent structure. This will definitely help in better variance estimation and subsequently, quicker convergence time.

ACKNOWLEDGEMENT

We would sincerely like to thank Romer Rosales, Rupesh Gupta, Ajith Muralidharan, Ajit Singh, Shaunak Chatterjee, Boyi Chen, Ian Ackerman and Dylan Wang for their detailed insightful feedback during the development and deployment of this system.

REFERENCES

[1] Deepak Agarwal, Bee-Chung Chen, Pradheep Elango, and Xuanhui Wang. 2011. Click Shaping to Optimize Multiple Objectives. In *KDD*. ACM, New York, NY, USA, 132–140.
[2] Deepak Agarwal, Bee-Chung Chen, Pradheep Elango, and Xuanhui Wang. 2012. Personalized Click Shaping Through Lagrangian Duality for Online Recommendation. In *SIGIR*. ACM, New York, NY, USA, 485–494.
[3] Deepak Agarwal, Bee-Chung Chen, Rupesh Gupta, Joshua Hartman, Qi He, Anand Iyer, Sumanth Kolar, Yiming Ma, Pannagadatta Shivaswamy, Ajit Singh, and Liang Zhang. 2014. Activity Ranking in LinkedIn Feed. In *KDD*. ACM, New York, NY, USA, 1603–1612.
[4] Deepak Agarwal, Bee-Chung Chen, Qi He, Zhenhao Hua, Guy Lebanon, Yiming Ma, Pannagadatta Shivaswamy, Hsiao-Ping Tseng, Jaewon Yang, and Liang Zhang. 2015. Personalizing LinkedIn Feed. In *KDD*. ACM, New York, NY, USA, 1651–1660.
[5] Shipra Agrawal and Navin Goyal. 2012. Analysis of Thompson Sampling for the Multi-armed Bandit Problem. In *Proceedings of the 25th Annual Conference on Learning Theory (Proceedings of Machine Learning Research)*, Shie Mannor, Nathan Srebro, and Robert C. Williamson (Eds.), Vol. 23. PMLR, Edinburgh, Scotland, 39.1–39.26.
[6] Shipra Agrawal and Navin Goyal. 2013. Thompson Sampling for Contextual Bandits with Linear Payoffs. In *ICML*. JMLR.org, USA, 1220–1228.
[7] Shipra Agrawal and Navin Goyal. 2017. Near-Optimal Regret Bounds for Thompson Sampling. *J. ACM* 64, 5 (Sept. 2017), 30:1–30:24.
[8] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning* 47, 2-3 (2002), 235–256.

[9] Kinjal Basu and Souvik Ghosh. 2017. Analysis of Thompson Sampling for Gaussian Process Optimization in the Bandit Setting. *arXiv:1705.06808* (2017).

[10] Hildo Bijl, Thomas B Schön, Jan-Willem van Wingerden, and Michel Verhaegen. 2016. A sequential Monte Carlo approach to Thompson sampling for Bayesian optimization. *arXiv:1604.00169* (2016).

[11] Fedor Borisyuk, Liang Zhang, and Krishnaram Kenthapadi. 2017. LiJAR: A System for Job Application Redistribution Towards Efficient Career Marketplace. In *KDD*. ACM, New York, NY, USA, 1397–1406.

[12] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press.

[13] Olivier Chapelle and Lihong Li. 2011. An Empirical Evaluation of Thompson Sampling. In *NIPS*. Curran Associates Inc., USA, 2249–2257.

[14] Josef Dick and Friedrich Pillichshammer. 2010. *Digital Nets and Sequences: Discrepancy Theory and Quasi-Monte Carlo Integration*. Cambridge University Press, New York, NY, USA.

[15] Ole-Christoffer Granmo. 2010. Solving two-armed Bernoulli bandit problems using a Bayesian learning automaton. *International Journal of Intelligent Computing and Cybernetics* 3, 2 (2010), 207–234.

[16] José Miguel Henrández-Lobato, Matthew W. Hoffman, and Zoubin Ghahramani. 2014. Predictive Entropy Search for Efficient Global Optimization of Black-box Functions. In *NIPS*. MIT Press, Cambridge, MA, USA, 918–926.

[17] Fred J. Hickernell. 1995. A Comparison of Random and Quasirandom Points for Multidimensional Quadrature. In *MCQMC*. Springer New York, New York, NY, 213–227.

[18] Donald R. Jones, Matthias Schonlau, and William J. Welch. 1998. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization* 13, 4 (01 Dec 1998), 455–492.

[19] Emilie Kaufmann, Olivier Cappe, and Aurelien Garivier. 2012. On Bayesian Upper Confidence Bounds for Bandit Problems. In *AISTATS*, Vol. 22. PMLR, La Palma, Canary Islands, 592–600.

[20] Benedict C. May, Nathan Korda, Anthony Lee, and David S. Leslie. 2012. Optimistic Bayesian Sampling in Contextual-bandit Problems. *J. Mach. Learn. Res.* 13 (June 2012), 2069–2106.

[21] Ha Quang Minh, Partha Niyogi, and Yuan Yao. 2006. Mercer's Theorem, Feature Maps, and Smoothing. In *Learning Theory*, Gábor Lugosi and Hans Ulrich Simon (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 154–168.

[22] Carl Edward Rasmussen and Christopher K. I. Williams. 2005. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

[23] R. Rivest. 1992. The MD5 Message-Digest Algorithm. (1992).

[24] Jacob Shekel. 1971. Test functions for multimodal search techniques. In *Fifth Annual Princeton Conf. on Information Science and Systems, 1971*.

[25] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *NIPS*. Curran Associates Inc., USA, 2951–2959.

[26] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. 2010. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *ICML*. Omnipress, USA, 1015–1022.

[27] Roshan Sumbaly, Jay Kreps, and Sam Shah. 2013. The Big Data Ecosystem at LinkedIn. In *SIGMOD*. ACM, New York, NY, USA, 1125–1134.

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

[29] Jarno Vanhatalo, Jaakko Riihimäki, Jouni Hartikainen, Pasi Jylänki, Ville Tolvanen, and Aki Vehtari. 2013. GPstuff: Bayesian Modeling with Gaussian Processes. *J. Mach. Learn. Res.* 14, 1 (April 2013), 1175–1179.