A Computationally Efficient On-Demand Experiment Platform

Thu Le
Lyft
San Francisco, CA, USA
thule@lyft.com

Lawrence R. De Geest
Lyft
San Francisco, CA, USA
ldegeest@lyft.com

1 Introduction

Randomized controlled trials (RCTs) or A/B tests were introduced in the 1920s [3] and are now standard in the technology industry to test the causal impact of product changes [4, 8, 10]. Modern A/B tests can consist of millions or even billions of subjects entering an experiment continuously in real-time and hundreds of metrics tested. Efficiently computing treatment effects and their variances is therefore crucial.

In this paper, we present the design and implementation of an experimentation platform that can accommodate large sample sizes, efficient and fast computations of the experiment outcomes, and advanced statistical analyses, both parametric such as Seemingly-Unrelated Regression (SUR) [7, 18], variance reduction through pre-experiment data (CUPED) [1], Heterogeneous Treatment Effect (HTE) [17], etc. and non-parametric such as Bayesian Bootstrapping [16]. We take advantage of the sparsity of the data and translate all the statistical analyses into sparse matrix computations to fit into memory. Our proposed experiment platform allows experimenters to more quickly interact with their results through a flexible on-demand analysis system to understand how treatment effects may vary across dimensions like geography and product type. We believe this is the first paper of its kind to outlines a detailed implementation of an efficient experimentation platform design that is extendable to almost all the often-requested sophisticated statistical analyses.

2 Anatomy of an Experimentation Platform

2.1 Components of an Experiment Platform

The principle of A/B testing is straightforward. Test subjects, for instance users (e.g., riders or drivers in the case of Lyft) or devices/browser cookies, are split randomly into multiple variants (e.g., app redesigns, matching algorithm improvements, targeted incentives), with the simplest experiment containing one treatment variant and one control, which retains the current status quo. Outcomes are then compared across variants to identify the lift or marginal impact of each variant over the control. There are a number of techniques available to optimize treatment allocation to reduce variance [11, 20] or minimize interference in complex settings such as marketplaces [6, 14, 15], but the general principle of randomization to ensure the validity of clean comparisons across variants remains unchanged.

From a platform point of view, there are three main components to an A/B test, captured in Figure 1:

(1) **Exposure Events ("Feature Flagging")**: When a user enters the experiment and hits a trigger point (e.g., entering a destination address to create a ride intent), the platform randomizes and delivers a variant assignment per the experiment's engineering specification. This assignment persists for the same user throughout the duration of the experiment (a "between-subjects" design). The platform then logs user_id, along with the timestamp that the user hits the trigger point, or the Exposure Event.

(2) **Metric Events**: The experimenter can specify any events or outcomes that they want to track over time from the users in the experiment (e.g., conversion, cancellations). The metric events exist independently of the feature flagging/exposure events.

(3) **Analysis Engine**: The platform then combines exposure events with metrics events to compare outcomes across variants. A variety of statistical techniques can be applied to measure impact, with the most basic being the student t-test [19] (equivalent to a linear model with an intercept and a treatment indicator) and to measure uncertainty in the treatment effect, including non-parametric methods such as bootstrapping [2] and permutation tests [3, 12, 13]. Variance reduction methods such as Cuped are deployed before estimating impacts to reduce uncertainty.

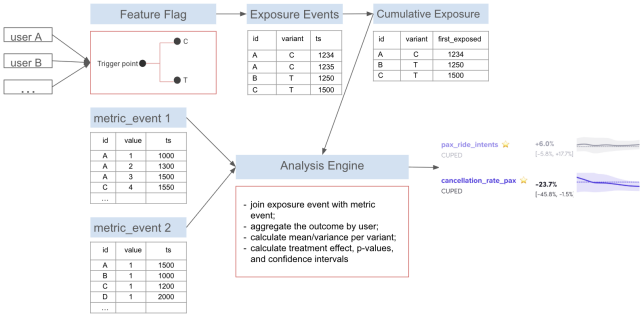

Figure 1: An experimentation platform: Exposure Events, Metric Events, and Analysis Engine

2.2 Challenges to the Standard Experimentation Platform

Building an in-house experimentation platform can facilitate discovery and streamline product improvements, but it can also be quite expensive in terms of data storage and computational resources. For starters, in a typical tech company, there might be hundreds or thousands of experiments running at the same time [5]. For each experiment, experimenters tend to be interested in hundreds of different metrics, which can overwhelm to Extract Transform Load (ETL) data pipelines.

In addition, discovering large impacts is challenging and most experiments lead to no-ship decisions [9], so it is crucial to get as much insight as possible from experiments to improve future experiments. This involves going beyond simple treatment-control comparisons to searching for HTE to find dimensions along which a treatment may have shown more movement than in the aggregate. Pre-calculating experiments results across all possible dimensions on a daily ETL pipeline is cost-prohibitive while an on-demand system requires several computational optimizations to deliver the results to the users at a reasonable wait-time.

Finally, estimating uncertainty in treatment effects using non-parametric methods like bootstrapping is preferred because they make fewer assumptions about the sample and population distributions of key metrics, but at the cost of expensive computation. For example, bootstrapping requires creating thousands of synthetic

datasets and performing the same intensive analysis thousands of times.

In the next section, we address these points and lay out the theoretical frameworks of matrix sparsity calculation and how, in practice, it can bring tremendous improvements to the system's runtime and memory. We will then discuss in detail how advanced statistical techniques like CUPED, HTE, ratio and incremental metric estimations using SUR, Bayesian Bootstrapping can be efficiently fit using a matrix sparsity framework.

3 Leveraging Data Sparsity

A key insight to our on-demand platform is that most user data is sparse or zero, which we can exploit in our analysis pipeline using efficient matrix operations. To demonstrate how matrix sparsity calculation can be applied in an experimentation platform, let us go through an example of a real-life experiment.

Suppose at Lyft we run a simple control/treatment experiment that randomizes riders upon having a ride-intent, i.e., when they enter the destination. Over the course of a week we collect 10 millions exposures, and we wish to estimate the treatment effect on metrics such as rides, cancellations, and so on. For each metric we would `LEFT OUTER JOIN` the 10M users in the experiment with the metric events table. This join retains all the users in the experiment, regardless of whether they recorded a metric event (for instance, users with no rides will take the value of 0). We then group by the `user_id` and calculate mean and variance per experiment arm accordingly. Figure 2 demonstrates this process.

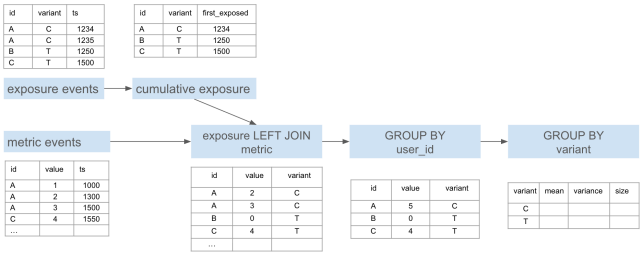

Figure 2: We left join exposure table with metric events and keep all the users who have been in the experiment regardless of whether they appear in the metric events or not.

The number of rows that we end up storing in this experiment becomes $m \times n$ with n being number of users in the experiment and m being number of metrics. So as both m and n increase, this join becomes untenable. Adding metric dimensions would significantly worsen the issue - each metric dimension would be treated as a separate metric, and m would grow substantially. For instance, suppose the material effect of a treatment is not a change in total number of rides, but instead a change in preferences across ride modes (e.g., Standard, Wait & Save, Luxury, etc.). In order to quantify this mode mix impact, we would need to create three separate metric values for each user and effectively triple the amount of rows needed to store under the usual `LEFT OUTER JOIN` method.

As mentioned above, most metrics are rather sparse - of all the users in the experiments, there would be a significantly smaller subset of users who would have non-zero metric values (rides, cancellations, etc.). When we break down a metric by dimension, for example, ride by different modes, the sparsity is even more noticeable - a very small number of users would have had Luxury rides in an experiment. Thus, the obvious solution would be to perform INNER JOIN exposure with metric events which effectively retains only the users who have non-zero metric values. Users who drop off from this join can be 'implicitly' understood to have zero value. This leads to the sparse matrix implementation, in which we treat the resulting inner join data as a Compressed Sparse Row (CSR) matrix. We then create index pointers to users with zero metric value.

To demonstrate the value of using sparse matrix calculations, we simulate treatment effect estimation in a high-dimensional setting where covariates reflect whether a rider took at least one ride in a given mode. Underlying ride counts are drawn from a Poisson distribution, whose rate parameter controls the degree of sparsity in the data. We then compare treatment effect estimation with a sparse least-squares solver (LSQR) to dense linear regression. In general we see that the value of sparse methods increase with both the size of the data and the sparsity of the data.

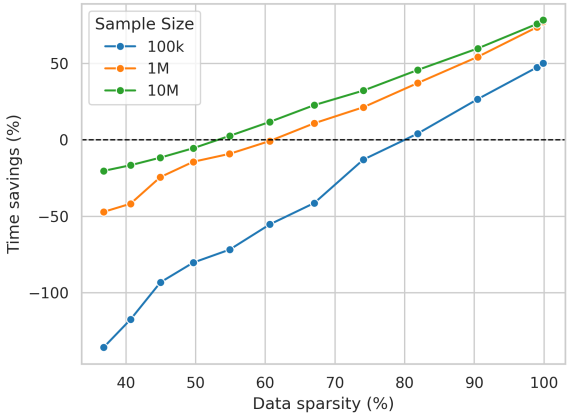

Figure 3: Time savings when estimating treatment effects using sparse versus dense methods on simulated rideshare data.

Storing experimental data in sparse matrix format saves considerable memory, and importantly, it does not preclude standard and advanced statistical analyses. In fact, given the significant memory savings, this approach will enable us to perform memory-intensive statistical tasks such as bootstrapping or HTE as demonstrated in the next section.

4 Analyzing Sparse Experiment Data

Suppose there are n users in the experiment with the outcome metric y. Let X denote the treatment indicator - in the two-variant case of treatment vs. control, $X = 1$ if user i is assigned to treatment and 0 otherwise (our framework easily extends to multiple variants). When testing the impact of the experiment on a sparse, cumulative metric like rides y we can estimate the linear model:

$$y = X\beta + \epsilon$$

using off-the-shelf sparse matrix packages (e.g., `scipy.sparse`) and easily extend it to estimate HTEs (e.g., the treatment effect of rides across regions or operating systems) by modifying the design matrix X. CUPED variance reduction is also easily done by adding pre-treatment covariates to X. This framework can be extended other experimental designs like within-subjects, in which a user experiences both treatment and control, by efficiently netting out the subject fixed effects before estimating the treatment effect, or by including random effects and other time-invariant factors like age. Depending on the temporal structure of the data, the variance-covariance matrix of β can easily be adjusted for clustering.

In addition, we can use regression to estimate more complex metrics with a numerator and denominator, like incremental metrics (e.g., cost-per-incremental-ride) and ratio metrics (e.g., conversion rate).

During SUR, we define two regression equations, one for numerator and one for denominator:

$$y_{num} = \alpha_{num} + \beta_{num} X + \epsilon_{num}$$
$$y_{den} = \alpha_{den} + \beta_{den} X + \epsilon_{den}$$

which are "seemingly unrelated" in that we allow information between the two models to flow through the error terms ϵ_{num} and ϵ_{den}. To estimate the joint effect we first estimate each model standalone, construct the variance-covariance matrix

$$\Sigma = \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{12} & \sigma_{22} \end{pmatrix}$$

and then estimate the treatment effects using Generalized Least Squares (GLS):

$$\hat{\beta}_{num,den} = \left(\mathbf{X}'(\Sigma^{-1} \otimes I_n)\mathbf{X} \right)^{-1} \mathbf{X}'(\Sigma^{-1} \otimes I_n)\mathbf{y}$$

The intuition behind this approach is that in an experiment, both numerator and denominator can be affected by shared latent factors. Taking conversion for instance, treatment might cause the user to open the app more (denominator) and also ride more (numerator). The SUR model allows for correlated error terms across the numerator and denominator equations and acknowledges that random variation in behavior affects both parts of the ratio or incremental metric. Moreover, because we obtain both variance and covariance terms between $\hat{\beta}_{num}$ and $\hat{\beta}_{den}$, we can easily calculate standard errors using the delta method.

The calculation depends on the metric at hand and simply involves a rearranging of the estimated coefficients. For an incremental metric we simply have

$$\mathbb{E}\left[\frac{y^T_{num} - y^C_{num}}{y^T_{den} - y^C_{den}} \right] = \frac{\hat{\beta}_{num}}{\hat{\beta}_{den}}.$$

For a ratio metric we can write expectations over the regression equations and obtain

$$E\left[\frac{y^T_{num}}{y^T_{den}} - \frac{y^C_{num}}{y^C_{den}} \right] = \frac{\hat{\alpha}_{num} + \hat{\beta}_{num}}{\hat{\alpha}_{den} + \hat{\beta}_{den}} - \frac{\hat{\alpha}_{num}}{\hat{\alpha}_{den}}.$$

Once more, this framework can be extended for within-subjects designs to account for user random effects and clustered errors as needed. In other words, our framework permits us to estimate treatment effects for a variety of metrics using generalized linear models and sparse matrices.

4.1 Bootstrap under Sparse Matrix Manipulation

Another advantage to our sparse-matrix platform is the ability to efficiently bootstrap highly-skewed metrics, or arbitrary metrics that combine treatment effects with other information like life-time values. By converting data into sparse formats, we can load sample sizes in the millions in memory, generate matrices of bootstrap weights using the Bayesian bootstrap framework [16], and quickly bootstrap empirical distributions. For example, suppose a team creates an arbitrary metric Y and wishes to bootstrap its distribution for different ride modes. Algorithm 1 lays out our process to inner join metrics and exposures, convert the resulting data into sparse matrices to preserve implicit zeros without storing them, and then Bayesian Bootstrap the empirical distribution for each mode.

Algorithm 1 Bayesian Bootstrap with Sparse Matrix Computation

```
R = INNER JOIN exposure E WITH metric events M
Extract sparse outcome matrix Y and treatment assignment vector X
for each treatment variant v do
    Construct sparse outcome matrices Y^T, Y^C from Y using X
    for each ride mode k, let B be the number of bootstraps desired do
        Draw Dirichlet weight matrix W^T ∈ R^{B×n_T} for treated users
        Draw Dirichlet weight matrix W^C ∈ R^{B×n_C} for control users
        Compute bootstrapped differences: τ̂^(1:B) = W^T Y^T_{:,k} − W^C Y^C_{:,k}
    end for
end for
Return empirical distribution {τ̂^(1:B)} for each treatment and ride mode
```

5 Conclusion

In this paper, we present a radically new computational framework for an experiment platform. The main contributions of this paper are 1.Lay out the general groundwork for implementing sparse matrix computation for analyzing experiment results; 2. Detail the solutions to solve the edge cases in the experiment analysis stage, specifically for CUPED, HTE, ratio and incremental metrics, and bootstrap. This work has paved the way for advanced statistical analyses in a real-time and interactive manner.

References

[1] Alex Deng, Ya Xu, Ron Kohavi, and Toby Walker. 2013. Improving the sensitivity of online controlled experiments by utilizing pre-experiment data. In *Proceedings of the sixth ACM international conference on Web search and data mining*. 123–132.

[2] Bradley Efron and Robert J Tibshirani. 1994. *An introduction to the bootstrap*. Chapman and Hall/CRC.

[3] Ronald Aylmer Fisher, Ronald Aylmer Fisher, Statistiker Genetiker, Ronald Aylmer Fisher, Statistician Genetician, Great Britain, Ronald Aylmer Fisher, and Statisticien Généticien. 1966. *The design of experiments*. Vol. 21. Springer.

[4] Somit Gupta, Ronny Kohavi, Diane Tang, Ya Xu, Reid Andersen, Eytan Bakshy, Niall Cardin, Sumita Chandran, Nanyu Chen, Dominic Coey, et al. 2019. Top challenges from the first practical online controlled experiments summit. *ACM SIGKDD Explorations Newsletter* 21, 1 (2019), 20–35.

[5] Somit Gupta, Lucy Ulanova, Sumit Bhardwaj, Pavel Dmitriev, Paul Raff, and Aleksander Fabijan. 2018. The anatomy of a large-scale experimentation platform. In *2018 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 1–109.

[6] David Holtz, Ruben Lobel, Inessa Liskovich, and Sinan Aral. 2020. Reducing interference bias in online marketplace pricing experiments. *arXiv preprint arXiv:2004.12489* (2020).

[7] Cindy D Kam and Marc J Trussler. 2017. At the nexus of observational and experimental research: Theory, specification, and analysis of experiments with heterogeneous treatment effects. *Political Behavior* 39 (2017), 789–815.

[8] Ron Kohavi, Alex Deng, Brian Frasca, Toby Walker, Ya Xu, and Nils Pohlmann. 2013. Online controlled experiments at large scale. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1168–1176.

[9] Ron Kohavi, Alex Deng, and Lukas Vermeer. 2022. A/B testing intuition busters: Common misunderstandings in online controlled experiments. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3168–3177.

[10] Ron Kohavi, Diane Tang, and Ya Xu. 2020. *Trustworthy online controlled experiments: A practical guide to a/b testing*. Cambridge University Press.

[11] Xiangrui Meng. 2013. Scalable simple random sampling and stratified sampling. In *International conference on machine learning*. PMLR, 531–539.

[12] Edwin JG Pitman. 1937. Significance tests which may be applied to samples from any populations. *Supplement to the Journal of the Royal Statistical Society* 4, 1 (1937), 119–130.

[13] Edwin James George Pitman. 1938. Significance tests which may be applied to samples from any populations: III. The analysis of variance test. *Biometrika* 29, 3/4 (1938), 322–335.

[14] Jean Pouget-Abadie, Vahab Mirrokni, David C Parkes, and Edoardo M Airoldi. 2018. Optimizing cluster-based randomized experiments under monotonicity. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2090–2099.

[15] Stephen W Raudenbush. 1997. Statistical analysis and optimal design for cluster randomized trials. *Psychological methods* 2, 2 (1997), 173.

[16] Donald B Rubin. 1981. The bayesian bootstrap. *The annals of statistics* (1981), 130–134.

[17] Sriram Somanchi, Ahmed Abbasi, Ken Kelley, David Dobolyi, and Ted Tao Yuan. 2023. Examining user heterogeneity in digital experiments. *ACM Transactions on Information Systems* 41, 4 (2023), 1–34.

[18] Virendera K Srivastava and David EA Giles. 2020. *Seemingly unrelated regression equations models: Estimation and inference*. CRC press.

[19] Student. 1908. The probable error of a mean. *Biometrika* (1908), 1–25.

[20] Xingwang Zhao, Jiye Liang, and Chuangyin Dang. 2019. A stratified sampling based clustering algorithm for large-scale data. *Knowledge-Based Systems* 163 (2019), 416–428.