Replication paper:

Detecting Interference in A/B Testing with Increasing Allocation

Vytautas Dominykas Leipus

Parametric and Nonparametric Statistics

Abstract

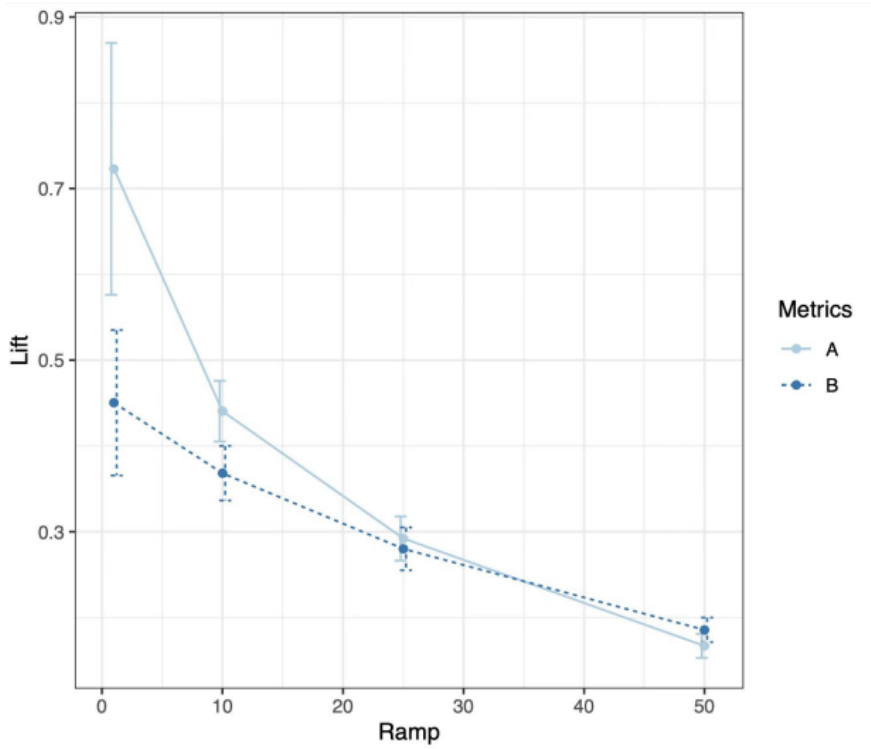
A/B testing and experimentation is currently common among companies offering services online, particularly social media. Ability to collect various data and related usage metrics from customers makes this experimentation design useful in solving if new implementations or already existing factors are causing the outcome. A possible and disrupting occurrence in this scenario is interference. Han et al. (2022) give a motivating example of an A/B test implemented by LinkedIn in which interference is a likely cause. To solve this issue, they propose a set of algorithms to test for interference in various cases. In this paper algorithms 1-3 from the study are replicated and compared to the original paper. Lastly, some considerations for implementing the algorithms are given.

Nowadays A/B testing and experimentation is widely used by various online services providers, especially social media, to decipher if newly implemented features or designs for clients help those providers get the desired outcome. Ability to collect various data and related usage metrics from customers makes this experimentation design useful in solving if new implementations or already existing factors are causing the outcome. However, one problem arises when in some cases no treatment given to a group or a subject can still indirectly cause the same outcome. This interference effect in many cases can be explained as some type of behavior change from treated subject that also affects the behavior of the untreated subject. In a study by Han et al. (2022) this problem is analyzed by using multiple algorithms that can be applied to an A/B testing scenario.

The motivating example given in the original paper studies an A/B test implemented by LinkedIn. In Figure 1 it is visible that difference-in-means estimator becomes smaller as more units are affected by treatment. Stable Unit Treatment Values Assumption (SUTVA) suggests that the difference in means should not change with increased treatment probability, however, in the given example it decreases. Authors suggest that one possible cause of this is interference.

Figure 1

An A/B test implemented by LinkedIn with increasing allocation.



Note. On the x-axis - the percentage of units that are in the treatment group; on the y-axis - the value of the difference-in-means estimator. Note that A and B stand for different outcome metrics. From Han, K., Li, S., Mao, J., & Wu, H. (2022). *Detecting Interference in A/B Testing with Increasing Allocation.* arXiv (Cornell University). doi:10.48550/arxiv.2211.03262

In this paper the goal is to replicate algorithms 1-3 described in the aforementioned paper using a different university social network data than in the original paper. The focus of this paper is spent more on replicating the algorithms through replicable code and analyzing the intuition and possible faults when applying the algorithms to real-life scenarios. Less focus is spent on results given that the experiments are only simulated, however, obtained p-values are to be compared with expected values in the original paper, to see if algorithms are performed correctly.

Data

For this replication paper the same Facebook100 dataset is used as in the original paper. The dataset is analyzed in detail by Traud et al. (2012). Additionally, the data is available for download (Social Structure of Facebook Networks Facebook Data Scrape, 2011). The dataset contains a set of friendships in Facebook social media platform at 100 US universities recorded at one point in time in 2005, moreover, it includes student data such as student/faculty status flag, gender, major, second

major/minor, dorm/house, year, and high school. The dataset contains a sparse matrix and a "*local_info*" variable, which presents all attributes for each student. For this analysis *local_info* was used to create a usable dataset. In this paper Yale network is analyzed, while in the original paper Swarthmore college network is used in simulations. All data manipulations are performed using R programming language and additional libraries used are specified in the code that can be seen in Appendix 6: Project code.

Execution of the logarithms

The first three algorithms give a universal framework for testing probability of interference. The first algorithm is constructed for a single experiment Appendix 3: Algorithm 1, 2nd algorithm describes a case when two experiments are plausible Appendix 4: Algorithm 2 and 3rd is a case for two experiments with introduction of time fixed effect Appendix 5: Algorithm 3. For all algorithms analyzed in the paper we take: repetitions of the algorithm – $B = 200$ and treatment probability – $\Pi = 50\%$.

The first algorithm can be perceived as the base model on which other algorithms are augmented. This case is designed for only one experiment, while following algorithms can be performed with data of two or more experiments. In certain real-world scenarios, only a single experiment will be plausible as a potential treatment on subjects can have an unrestorable effect on the subject and its related variables, thus affecting how subjects interact with treatment in repeated experiments. On the other hand, the original paper focuses on social media and online marketplaces which are inherently dynamic and create possibilities for multiple experiments. Additionally, time between experiments can have an impact on outcome and thus making testing for interference more difficult, however, algorithm 3 accounts for this problem using a time fixed effect assumption.

Algorithm 1. Given how the 1st algorithm is constructed, multiple steps of data manipulation are required. One of the first important steps is to generate a Bernoulli random variable that will represent the simulated binary treatment variable *W*. This is performed by using "*rbinom*" function in R and adding additional column to the dataset. Another additional variable H that denotes the treatment of neighbors of the candidate is calculated as simply the sum of neighbors that were lucky to get the treatment except the candidate itself. Code for these steps is shown in Figure 2.

Figure 2

```
# Adding a new column/variable to the dataset using bernoulli randomisation.
# This variable represents the binary treatment value.
# pi = 0.5 or 50%
datafr$W <- c(rbinom(8578,1,0.5))

# Adding/Creating candidate exposure function variable in the dataset
datafr$H<-NA
datafr[datafr$W==1,"H"] <- sum(datafr$W ==1)-1
datafr[datafr$W==0,"H"] <- sum(datafr$W ==1)
```

After having divided the dataset into focal and auxiliary units as described in the original paper (this step can be observed in the Appendix 6: Project code), next step is to obtain the coefficient from a regression model for *H* of the focal units. Variable *V3* indicating major is chosen as dependent variable. Important to note that variable *W* was excluded from the regression model due to having a direct linear relationship with *H*, causing multicollinearity. How coefficient is obtained from regression model is shown in Figure 3.

Figure 3

```
# Alg. 1.2.
# Regression for T - W is excluded as it has a linear relationship with H
Obtain_T = lm(V3 ~V1 + V2 + V4 + V5 + V6 + V7 + H, data = Data_foc)
summary(Obtain_T)

# Naming and saving the coefficient for further calculations
H_0 <- Obtain_T$coefficients[8]
```

In Figure 4 we can see the main loop for obtaining the p-value. Firstly, the binary treatment variable W is updated by regenerating it using the same Bernoulli randomization as in initial generation. Then we recalculate candidate exposure function because it depends on the number of treated candidates in the dataset. With updated values we perform a linear regression to obtain a coefficient to compare it with the initial coefficient obtained in previous steps. To compare the coefficients an indicator function within the loop is created. In this case it is a simple conditional statement which increases the sum of *i* by one for every time the hypothesis is not rejected when comparing the coefficients. Lastly, the p-value is calculated using the formula denoted in the algorithm and using the calculated sum of *i* as input.

Figure 4

```
B=200

for (i in 1:B)
{
  Data2_aux$W <- c(rbinom(4289,1,0.5)) # 1.3.1 Regenerating treatments...
  Data1_foc$H<-NA # 1.3.2 recomputing the candidate exposure...
  Data1_foc[Data1_foc$W==1,"H"] <- sum(Data1_foc$W ==1)+sum(Data2_aux$W ==1)-1
  Data1_foc[Data1_foc$W==0,"H"] <- sum(Data1_foc$W ==1)+sum(Data2_aux$W ==1)
  Tb = lm(V3 ~V1 + V2 + V4 + V5 + V6 + V7 + H, data = Data1_foc) # 1.3.3
  summary(Tb) # 1.3.3
  H_b <- Tb$coefficients[8]
  if (H_0<=H_b) # for sum of indicator function notating T0<=Tb
  {
    i=i+1
  }
}
# Output
P=(1/(B+1))*(1+i)
P
```

Algorithm 2. The 2nd algorithm introduces an additional experiment on the same data**.** The steps for the code are shown in Figure 5. First of all, we name the two datasets from the main data to represent two experiments. Similarly, as in algorithm 1, we need to generate the binary treatment variable and candidate exposure function for each experiment. The code part is nearly identical to the one in algorithm 1.

Figure 5

```
# Algorithm 2.
# Data for 2 experiments
dset1 <- as.data.frame(data1$local.info)
dset2 <- as.data.frame(data1$local.info)

# Generating treatments for 2 experiments
# and calculating exposure function for each candidate
dset1$W <- c(rbinom(8578,1,0.5))
View(dset1)

dset1$H<-NA
dset1[dset1$W==1,"H"] <- sum(dset1$W ==1)-1
dset1[dset1$W==0,"H"] <- sum(dset1$W ==1)

dset2$W <- c(rbinom(8578,1,0.5))
View(dset2)

dset2$H<-NA
dset2[dset2$W==1,"H"] <- sum(dset2$W ==1)-1
dset2[dset2$W==0,"H"] <- sum(dset2$W ==1)
```

As in this case we observe data for two experiments, consequently, we have four possible options of candidates experiencing or not experiencing treatment. We have two groups that either never had treatment applied between the two experiments or had it both times. Other two groups experienced treatment in either 1st or 2nd experiment. In this case we are interested only in the first two groups, the groups that had the same W value either 0 or 1 in both experiments. To complete this step treatment and candidate exposure variables denoted as *W2* and *H2* were added to the dataset of the first experiment – dset1. This step will help with sub-setting the dataset in to groups that had the same treatment between experiments. Additionally, the difference *Ydiff* for outcome variable *V3* between experiments is calculated. Calculating *Ydiff* for each subject instead of using separate *Y's* reduces the variance in the test statistic by removing variance that is shared between the outcome variables of both experiments. Lastly, we finally sample the two groups with identical value of the treatment variable between the two experiments. The mention steps are shown in Figure 6.

Figure 6

```
# 2.1. Creating the set of units whose treatment didn't change over the experiments
# Creating additional variables to include k2 variables in k1 dataset for simplicity
dset1$W2 <-NA
dset1$W2 <- dset2$W
dset1$H2 <-NA
dset1$H2 <- dset2$H

# Calculating Ydiff for the dataset. It will be used later on in 2.2.
Ydiff <- dset2$V3-dset1$V3
dset1$Ydiff<-NA
dset1$Ydiff<- Ydiff_foc

# Ind_nc - Set of units whose treatment didn't change over the experiments
Ind_nc <- subset(dset1, dset1$W==dset1$W2)

Sr <- sample(1:nrow(Ind_nc), nrow(Ind_nc)/2, replace = F)

Ind_foc <- Ind_nc[Sr, ]
Ind_aux <- Ind_nc[-c(Sr), ]
```

Along with outcome differences we need exposure function differences *Hdiff* for each candidate to use in regression model. Similarly, as in algorithm 1, in the next step we perform a regression model to obtain the coefficient, although it has additional variables to account for two experiments. In this case the final coefficient obtained is *H* instead of *Hdiff* because due to how the exposure function is calculated the difference becomes a constant number for each *i* that has no explanatory power on the dependent variable. Aforementioned steps are shown in Figure 7.

Figure 7

```
# Creating additional Hdff=H2-H1 variable,
# to express differences of candidate exposure function between experiments.
# This variable is to be used in regression model
# to obtain the coefficient for further calculations.
Ind_foc$Hdiff<-NA
Ind_foc$Hdiff<-Ind_foc$H2-Ind_foc$H

# 2.2.
Alg2Obtain_T = lm(Ydiff ~V1 + V2 + V4 + V5 + V6 + V7 + H + Hdiff , data = Ind_foc)
summary(Alg2Obtain_T)
Alg2_T0 <- Alg2Obtain_T$coefficients[8]
```

Lastly, we perform the for loop that compares the coefficients to test the hypothesis. Main difference compared to previous algorithm, is that instead of regenerating treatments we permute them between rows. Illustration for the algorithm can be seen in Appendix 1: An illustration of Algorithm 2. This is explained in the original paper as a guarantee that the procedure is valid because in this case choice of focal units depend on the treatment and this step is needed to obtain an accurate p-value. The for loop is visible in Figure 8.

Figure 8

```
# 2.3.
for (j in 1:B)
{
  # 2.3.1 Permutate treatments - permutate rows
  # For W
  perm1<-as.data.frame(Ind_aux1$W)
  perm1<- perm1 %>% sample_n(nrow(.))
  Ind_aux1$W<-perm1
  # For W2
  perm2<-as.data.frame(Ind_aux1$W2)
  perm2<- perm2 %>% sample_n(nrow(.))
  Ind_aux1$W2<-perm2
  # 2.3.2
  # Recompute candidate exposure Hfoc for k1 and k2, or H and H2 in the combined dataset
  Ind_foc1$H<-NA
  Ind_foc1[Ind_foc1$W==1,"H"] <- sum(Ind_foc1$W ==1)+sum(Ind_aux1$W ==1)-1
  Ind_foc1[Ind_foc1$W==0,"H"] <- sum(Ind_foc1$W ==1)+sum(Ind_aux1$W ==1)

  Ind_foc1$H2<-NA
  Ind_foc1[Ind_foc1$W2==1,"H2"] <- sum(Ind_foc1$W2 ==1)+sum(Ind_aux1$W2 ==1)-1
  Ind_foc1[Ind_foc1$W2==0,"H2"] <- sum(Ind_foc1$W2 ==1)+sum(Ind_aux1$W2 ==1)
  # 2.3.3
  Tb_Alg2 = lm(Ydiff ~V1 + V2 + V4 + V5 + V6 + V7 + H + Hdiff, data = Ind_foc1) # 1.3.3
  summary(Tb_Alg2)
  H_b_Alg2 <- Tb_Alg2$coefficients[8]
  if (Alg2_T0<=H_b_Alg2) # for sum of indicator function notating T0<=Tb
  {
    j=j+1
  }
}
Alg2_P=(1/(B+1))*(1+j)
Alg2_P
```

Algorithm 3. In Algorithm 3 we have an addition of assuming a time-fixed effect. The code steps shown in Figure 9 subset data in two groups based on value of binary variable *W*, either *W*=0 or *W*=1. Then the data is arranged by ascending order by year variable *V6* as a form of semi random matching. Some sort of matching algorithm can be performed to have a better match and reduce variance based on more covariates. Potential matching methods for this case can be explored in a study by Stuart (2010), although it is not the focus of this paper. However, authors state that even with random matching the test can remain valid. Lastly, additional datasets *Ind_0l* and *Ind_1l* are notated in code to use in code for clarity. This step is not necessary and subsets *Ind_0* and *Ind_1* can be used in the for loop as well.

Figure 9

```
# Algorithm 3.
# Using the same 2 experiment data used in algorithm 2
#dset1, dset2
# 3.1.
# Subsetting data from both experiments in to two groups - W=0 and W=1
Ind_0 <- subset(dset1, dset1$W==dset1$W2 & dset1$W==0)
Ind_1 <- subset(dset1, dset1$W==dset1$W2 & dset1$W==1)
# 3.2.
# Arranging data by year as a form of semi-random matching.
Ind_0 <- arrange(Ind_0,Ind_0$V6)
Ind_1 <- arrange(Ind_1,Ind_1$V6)
Ind_0 <- sample_n(Ind_0,nrow(Ind_1)) # making the data groups same row length

Ind_0l <- Ind_0
Ind_1l <- Ind_1
```

In Figure 10 we see more steps of algorithm 3. In these steps we calculate difference of outcomes between matched indices and compute a test statistic. Authors of the study note that difference-in-differences statistic is also plausible to obtain the *T* score, thus it was chosen in this case instead of a regression coefficient for simplicity.

Figure 10

```
# 3.3.
# Computing Ydiff_Ind1 and Test statistic
Ind_1$Ydiff_Ind1<-NA
Ind_1$Ydiff_Ind1<- Ind_1$V3-Ind_0$V3

# Computing an alternative test statistic
# Subtracting the same Ydiff as covariates between experiments were unchanged
Alg3_T0 <- mean(Ind_1$Ydiff_Ind1)-mean(Ind_1$Ydiff_Ind1)
```

In Figure 11 we can see the final loop with a nested loop inside for calculating the p-value. Firstly, we focus on the nested loop within the whole loop. The goal is to randomly permute outcomes between experiments or time points. The reasoning is that outcome differences between matched units and experiments should have the same distribution and perhaps even equal and thus permuting the outcomes should not affect it. If that is not the case, and means of outcome differences are different, there might be evidence for interference. A visual representation of this algorithm can be seen in Appendix 2: An illustration of Algorithm 3. The nested loop utilizes a Bernoulli randomized variable as input to a conditional statement which then either permutes outcomes or leaves them as they are. In the rest of the for loop we recalculate the difference of outcomes and recompute the test statistic to compare with the initial statistic. Authors of the study note that difference-in-differences statistic is

also plausible to obtain the *T* score, thus it was chosen in this case instead of a regression coefficient for simplicity.

Figure 11

```
# 3.4.
rowsl<-nrow(Ind_1l)
# For loop 3.4.
for (k in 1:B)
{
  # Nested loop for random permutation - binomial randomization with probability 0.5
  for (m in 1:rowsl)
  {
    binom_perm <- rbinom(1, size=1, prob=0.5)
    if (binom_perm==1)
    {
      # Outcomes across experiments are identical in this case,
      # thus permutating only for outcomes of matched groups/indices.
      # In real case scenario when outcomes differ,
      # an identical algorithm can be used for difference between experiments.
      # a1 & b1 as mediator variables
      # to avoid duplication of variables between experiments
      a1<-Ind_1l$V3[m]
      b1<-Ind_0l$V3[m]
      Ind_1l$V3[m]<-b1
      Ind_0l$V3[m]<-a1
    }}
  # Recomputing Ydiff
  Ind_1l$Ydiff_Ind1<- NA
  Ind_1l$Ydiff_Ind1<- Ind_1l$V3-Ind_0l$V3

  # Recomputing test statistic - difference in means
  Alg3_Tb <- mean(Ind_1l$Ydiff_Ind1)-mean(Ind_1l$Ydiff_Ind1)
  if (Alg3_T0<=Alg3_Tb) # for sum of indicator function notating T0<=Tb
  {
    k=k+1
  }
}
Alg3_P=(1/(B+1))*(1+k)
Alg3_P
```

Results and considerations

Given that the experiment is simulated and the data is used to only perform the logarithms and not inference, we can still analyze the obtained p-values from the algorithms and see how they compare to expected outcomes in the original paper. For all three algorithms the p-value obtained is 1. In the original paper the authors note that p-value under the null hypothesis can be stochastically larger than Unif[0,1] for the algorithm 1. Also, the authors note that for algorithm 3 $T^{(0)}$ and $T^{(b)}$ will have different distributions and p-value will be far from the Unif[0,1] distribution because after permutations

difference-in-differences statistic will have a mean zero. Lastly, p-values depend on the hypothesis and given a different hypothesis obtained p-values could also be completely opposite.

One consideration mentioned in the original paper is that p-values could be aggregated. Splitting of data or random matching can introduce randomness to the p-value. Simple solution to lessen this effect would be to run the algorithms multiple times and out of aggregated p-values create and aggregated test statistic.

Another consideration is how the candidate exposure function H is constructed. It appears that using it simply the way it is described can bring multicollinearity problems if coefficient is to be obtained from a regression rather than difference-in-differences test statistic. One way to solve this problem would be to create a candidate exposure function, which would not have a direct linear relationship with the treatment variable.

Additionally, it is important to mention that in a real experiment some covariates are likely to be affected by treatment. In this paper indexing of groups in the code part was simply done by combining data of one group with differing H and W from another group to match as an index. In case where covariates are different for the same subject in different group, it would be better to index or match the subjects by subject id or other identificatory. Moreover, for algorithm 3 and the matching step, it would be wise to experiment on matching algorithms to further reduce variance and increase power of the test.

Algorithms 4-5 that describe cases with multiple experiments and time-fixed effect assumption were excluded from this paper. The approach is not too different from algorithms 1-3. Authors of the study note that algorithm 3 is most similar to algorithm 5 and could be expanded to multiple experiments with an assumed time fixed effect model, however algorithms 4-5 are purely intended for multiple experiments which helps increasing power of the test.

References

Han, K., Li, S., Mao, J., & Wu, H. (2022). *Detecting Interference in A/B Testing with Increasing Allocation.* arXiv (Cornell University). doi:10.48550/arxiv.2211.03262

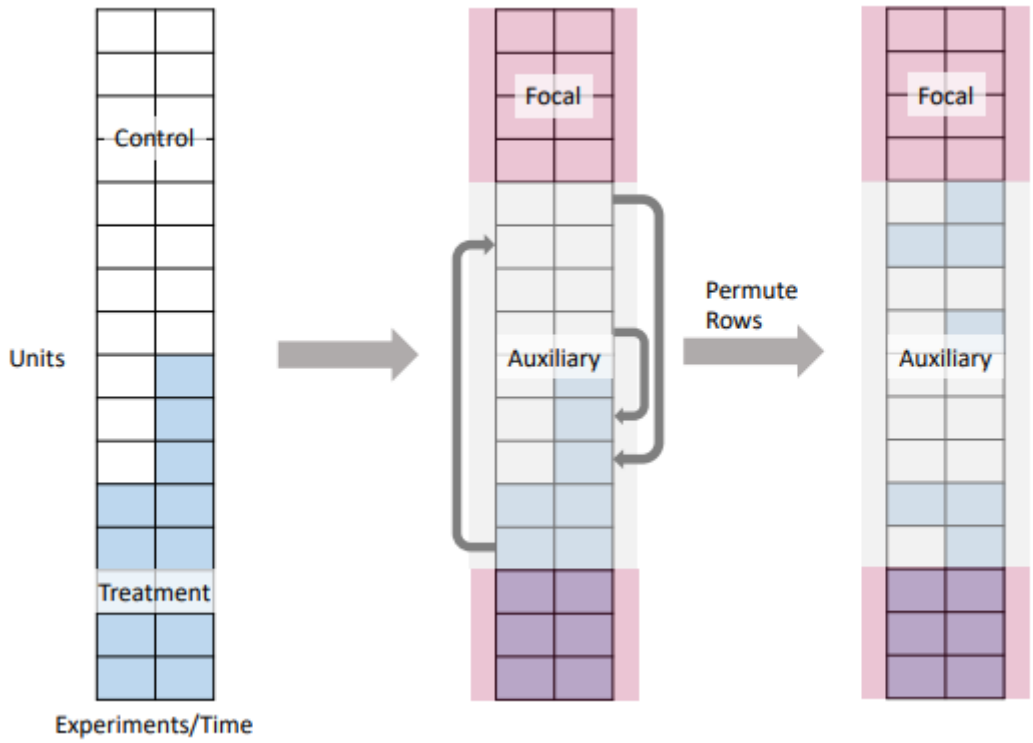
Social Structure of Facebook Networks Facebook Data Scrape. (2011). Retrieved from Internet Archive: https://archive.org/details/oxford-2005-facebook-matrix

Stuart, E. A. (2010, February 1). Matching methods for causal inference: A review and a look forward. *Statistical Science, 25*(1), 1–21. doi:10.1214/09-STS313

Traud, A. L., Mucha, P. J., & Porter, M. A. (2012). Social Structure of Facebook Networks. *Physica A, 391*(16), 4165-4180. doi:10.2139/ssrn.1470768

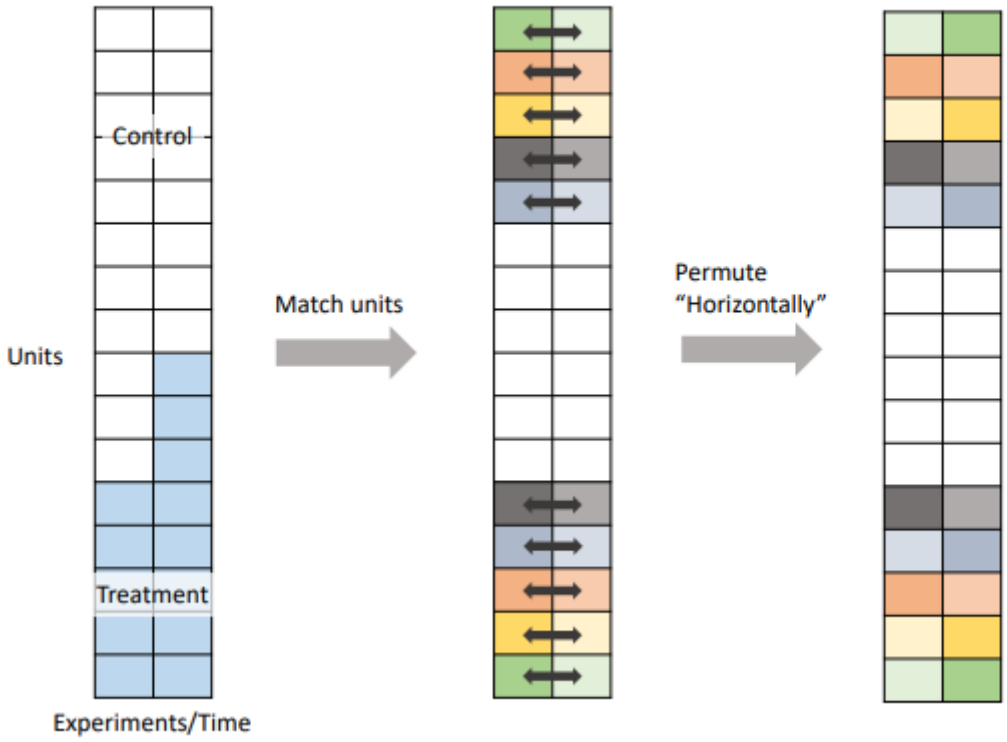
Appendix

Appendix 1: An illustration of Algorithm 2.



Note. After selecting the set of focal units and auxiliary units, we randomly permute rows of the treatment matrix and compute test statistics and p-values based on the permuted data. From Han, K., Li, S., Mao, J., & Wu, H. (2022). *Detecting Interference in A/B Testing with Increasing Allocation.* arXiv (Cornell University). doi:10.48550/arxiv.2211.03262

Appendix 2: An illustration of Algorithm 3.



Note: Algorithm 3 permutes the outcomes horizontally (across experiments), whereas Algorithm 2 permutes the treatments vertically (across units). From Han, K., Li, S., Mao, J., & Wu, H. (2022). Detecting Interference in A/B Testing with Increasing Allocation. arXiv (Cornell University). doi:10.48550/arxiv.2211.03262

Appendix 3: Algorithm 1

Algorithm 1 Testing for interference effect (one experiment).

Input: Dataset $\mathcal{D} = (W_{1:n}, X_{1:n}, Y_{1:n}, H_{1:n})$, exposure function h, test statistic T.

1. Randomly split the data into two folds. Let $\mathcal{I}_{\text{foc}}$ and $\mathcal{I}_{\text{aux}}$ be the index set for the first fold (focal units) and the second fold (auxiliary units). Write the first fold of data as $\mathcal{D}_{\text{foc}} = (W_{\text{foc}}, X_{\text{foc}}, Y_{\text{foc}}, H_{\text{foc}})$ and the second as $\mathcal{D}_{\text{aux}} = (W_{\text{aux}}, X_{\text{aux}}, Y_{\text{aux}}, H_{\text{aux}})$.
2. Compute a test statistic $T^{(0)} = T(W_{\text{foc}}, X_{\text{foc}}, Y_{\text{foc}}, H_{\text{foc}})$ that captures the importance of H in predicting Y.
3. **For** $b = 1, \dots B$:

   Regenerate treatments for the auxiliary units: $\widetilde{W}_i^{(b)} \sim \text{Bernoulli}(\pi)$ for $i \in \mathcal{I}_{\text{aux}}$.

   Recompute the candidate exposure for focal units: $\widetilde{H}_i^{(b)} = h_i(W_{\text{foc}\setminus\{i\}}, \widetilde{W}_{\text{aux}}^{(b)})$ for $i \in \mathcal{I}_{\text{foc}}$.

   Recompute the test statistic: $T^{(b)} = T(W_{\text{foc}}, X_{\text{foc}}, Y_{\text{foc}}, \widetilde{H}_{\text{foc}}^{(b)})$.

   End For

Output: The p-value

$$p = \frac{1}{B+1}\left(1 + \sum_{b=1}^{B} \mathbb{1}\left\{T^{(0)} \le T^{(b)}\right\}\right). \tag{4}$$

Note. From Han, K., Li, S., Mao, J., & Wu, H. (2022). Detecting Interference in A/B Testing with Increasing Allocation. arXiv (Cornell University). doi:10.48550/arxiv.2211.03262

Appendix 4: Algorithm 2

Algorithm 2 Testing for interference effect (two experiments).

Input: Datasets $\mathcal{D}_1 = (W_{1:n,1}, X_{1:n}, Y_{1:n,1}, H_{1:n,1})$, $\mathcal{D}_2 = (W_{1:n,2}, X_{1:n}, Y_{1:n,2}, H_{1:n,2})$, exposure function h, test statistic T.

1. Let $\mathcal{I}_{\text{nc}} = \{i : W_{i,1} = W_{i,2}\}$ be the set of units whose treatment didn't change over the experiments. Randomly sample a subset of $\mathcal{I}_{\text{nc}}$ of size $n/2$. We call the subset $\mathcal{I}_{\text{foc}}$. Let $\mathcal{I}_{\text{aux}} = [n] \setminus \mathcal{I}_{\text{foc}}$.
2. Take the difference of $Y_{\text{foc},2}$ and $Y_{\text{foc},1}$: let $Y^{\text{diff}}_{\text{foc}} = Y_{\text{foc},2} - Y_{\text{foc},1}$. Compute a test statistic $T^{(0)} = T(W_{\text{foc},1:2}, X_{\text{foc}}, Y^{\text{diff}}_{\text{foc}}, H_{\text{foc},1:2})$ that captures the importance of H in predicting Y^{diff}.
3. **For** $b = 1, \dots B$:

   Randomly permute treatments for the auxiliary units of the data: $\widetilde{W}^{(b)}_{i,1:2} = W_{\sigma^{(b)}(i),1:2}$ for $i \in \mathcal{I}_{\text{aux}}$, for some permutation $\sigma^{(b)}$ of $\mathcal{I}_{\text{aux}}$.

   Recompute the candidate exposure for the focal units: $\widetilde{H}^{(b)}_{i,k} = h_i(W_{\text{foc}\setminus\{i\},k}, \widetilde{W}^{(b)}_{\text{aux},k})$ for $i \in \mathcal{I}_{\text{foc}}$ and $k \in \{1, 2\}$.

   Recompute the test statistic: $T^{(b)} = T(W_{\text{foc},1:2}, X_{\text{foc}}, Y^{\text{diff}}_{\text{foc}}, \widetilde{H}^{(b)}_{\text{foc},1:2})$.

   End For

Output: The p-value

$$p = \frac{1}{B+1}\left(1 + \sum_{b=1}^{B} \mathbb{1}\left\{T^{(0)} \le T^{(b)}\right\}\right). \tag{5}$$

Note. From Han, K., Li, S., Mao, J., & Wu, H. (2022). Detecting Interference in A/B Testing with Increasing Allocation. arXiv (Cornell University). doi:10.48550/arxiv.2211.03262

Appendix 5: Algorithm 3

Algorithm 3 Testing for interference effect (two experiments, time fixed effect model).

Input: Datasets $\mathcal{D}_1 = (W_{1:n,1}, X_{1:n}, Y_{1:n,1}, H_{1:n,1})$, $\mathcal{D}_2 = (W_{1:n,2}, X_{1:n}, Y_{1:n,2}, H_{1:n,2})$, matching algorithm m, test statistic T.

1. Let $\mathcal{I}_0 = \{i : W_{i,1} = W_{i,2} = 0\}$ and $\mathcal{I}_1 = \{i : W_{i,1} = W_{i,2} = 1\}$.
2. For each i in $\mathcal{I}_1$, match an index $j \in \mathcal{I}_0$ to i (with no repeat): let $m(i)$ be the matched index of i. Let $\mathcal{I}_m = \{m(i) : i \in \mathcal{I}_1\}$ be the set of matched indices.[4]
3. For each $k \in \{1, 2\}$, compute $Y^{\text{diff}}_{\mathcal{I}_1,k} = \left(Y_{i,k} - Y_{m(i),k}\right)_{i\in\mathcal{I}_1}$, which is the vector of differences between the outcomes of the treated units and those of the matched units.
   Compute a test statistic $T^{(0)} = T(Y^{\text{diff}}_{\mathcal{I}_1,1:2}, X_{\mathcal{I}_m}, H_{\mathcal{I}_m,1:2}, X_{\mathcal{I}_1}, H_{\mathcal{I}_1,1:2})$.
4. **For** $b = 1, \dots B$:
   For each $i \in \mathcal{I}_1$:
   Randomly permute outcomes across experiments: $\widetilde{Y}^{(b)}_{i,k} = Y_{i,\sigma_{i,b}(k)}$ and $\widetilde{Y}^{(b)}_{m(i),k} = Y_{m(i),\sigma_{i,b}(k)}$ for some permutation $\sigma_{i,b}$ of $\{1, 2\}$.
   End For
   Recompute $\widetilde{Y}^{\text{diff},(b)}_{\mathcal{I}_1,k} = (\widetilde{Y}^{(b)}_{i,k} - \widetilde{Y}^{(b)}_{m(i),k})_{i\in\mathcal{I}_1}$.
   Recompute the test statistic: $T^{(b)} = T(\widetilde{Y}^{\text{diff}(b)}_{\mathcal{I}_1,1:2}, X_{\mathcal{I}_m}, H_{\mathcal{I}_m,1:2}, X_{\mathcal{I}_1}, H_{\mathcal{I}_1,1:2})$.
   End For

Output: The p-value

$$p = \frac{1}{B+1}\left(1 + \sum_{b=1}^{B} \mathbb{1}\left\{T^{(0)} \le T^{(b)}\right\}\right). \tag{9}$$

Note. From Han, K., Li, S., Mao, J., & Wu, H. (2022). Detecting Interference in A/B Testing with Increasing Allocation. arXiv (Cornell University). doi:10.48550/arxiv.2211.03262

Appendix 6: Project code

```
## Reading and previewing code

library(R.matlab)

library(tidyverse)

library(dplyr)

data1 <- readMat("C:/Users/Vartotojas/Desktop/PS Project
data/facebook100/Yale4.mat")


## Using detailed data instead of the sparse matrix that is also in the dataset.

data1$local.info

View(data1$local.info)
```

```
# Creating a separate dataframe of the dataset that will be used.
datafr <- as.data.frame(data1$local.info)
datafr

# Adding a new column/variable to the dataset using bernoulli randomisation.
# This variable represents the binary treatment value.
# pi = 0.5 or 50%
datafr$W <- c(rbinom(8578,1,0.5))

# Adding/Creating candidate exposure function variable in the dataset
datafr$H<-NA
datafr[datafr$W==1,"H"] <- sum(datafr$W ==1)-1
datafr[datafr$W==0,"H"] <- sum(datafr$W ==1)

# Calculating rows to specify sampling: rows - 8578, rows/2=4289
nrow(datafr)

# Alg. 1.1.
# Splitting data to focal and auxiliary units.
Sample_rows <- sample(1:nrow(datafr), 4289, replace = F)

Data_foc <- datafr[Sample_rows, ]
Data_aux <- datafr[-c(Sample_rows), ]

Data1_foc<-Data_foc # not necessary
Data2_aux<-Data_aux # not necessary

# Alg. 1.2.
# Regression for T - W is excluded as it has a linear relationship with H
Obtain_T = lm(V3 ~V1 + V2 + V4 + V5 + V6 + V7 + H, data = Data_foc)
summary(Obtain_T)
```

```
# Naming and saving the coefficient for further calculations
H_0 <- Obtain_T$coefficients[8]

B=200

for (i in 1:B)
{
  Data2_aux$W <- c(rbinom(4289,1,0.5)) # 1.3.1 Regenerating treatments...
  Data1_foc$H<-NA # 1.3.2 recomputing the candidate exposure...
  Data1_foc[Data1_foc$W==1,"H"] <- sum(Data1_foc$W ==1)+sum(Data2_aux$W ==1)-1
  Data1_foc[Data1_foc$W==0,"H"] <- sum(Data1_foc$W ==1)+sum(Data2_aux$W ==1)
  Tb = lm(V3 ~V1 + V2 + V4 + V5 + V6 + V7 + H, data = Data1_foc) # 1.3.3
  summary(Tb) # 1.3.3
  H_b <- Tb$coefficients[8]
  if (H_0<=H_b) # for sum of indicator function notating T0<=Tb
  {
    i=i+1
  }
}
# Output
P=(1/(B+1))*(1+i)
P

# Algorithm 2.
# Data for 2 experiments
dset1 <- as.data.frame(data1$local.info)
dset2 <- as.data.frame(data1$local.info)

# Generating treatments for 2 experiments
# and calculating exposure function for each candidate
dset1$W <- c(rbinom(8578,1,0.5))
```

```
View(dset1)


dset1$H<-NA

dset1[dset1$W==1,"H"] <- sum(dset1$W ==1)-1

dset1[dset1$W==0,"H"] <- sum(dset1$W ==1)


dset2$W <- c(rbinom(8578,1,0.5))

View(dset2)


dset2$H<-NA

dset2[dset2$W==1,"H"] <- sum(dset2$W ==1)-1

dset2[dset2$W==0,"H"] <- sum(dset2$W ==1)


# 2.1. Creating the set of units whose treatment didn't change over the
experiments

# Creating additional variables to include k2 variables in k1 dataset for
simplicity

dset1$W2 <-NA

dset1$W2 <- dset2$W

dset1$H2 <-NA

dset1$H2 <- dset2$H


# Calculating Ydiff for the dataset. It will be used later on in 2.2.

Ydiff <- dset2$V3-dset1$V3

dset1$Ydiff<-NA

dset1$Ydiff<- Ydiff_foc


# Ind_nc - Set of units whose treatment didn't change over the experiments

Ind_nc <- subset(dset1, dset1$W==dset1$W2)


Sr <- sample(1:nrow(Ind_nc), nrow(Ind_nc)/2, replace = F)


Ind_foc <- Ind_nc[Sr, ]
```

```
Ind_aux <- Ind_nc[-c(Sr), ]

# Additional variables to use in for loop (not necessary, for clarity)
Ind_foc1 <- Ind_foc
Ind_aux1 <- Ind_aux

# Creating additional Hdff=H2-H1 variable,
# to express differences of candidate exposure function between experiments.
# This variable is to be used in regression model
# to obtain the coefficient for further calculations.
Ind_foc$Hdiff<-NA
Ind_foc$Hdiff<-Ind_foc$H2-Ind_foc$H

# 2.2.
Alg2Obtain_T = lm(Ydiff ~V1 + V2 + V4 + V5 + V6 + V7 + H + Hdiff , data =
Ind_foc)
summary(Alg2Obtain_T)
Alg2_T0 <- Alg2Obtain_T$coefficients[8]

# 2.3.
for (j in 1:B)
{
  # 2.3.1 Permutate treatments - permutate rows
  # For W
  perm1<-as.data.frame(Ind_aux1$W)
  perm1<- perm1 %>% sample_n(nrow(.))
  Ind_aux1$W<-perm1
  # For W2
  perm2<-as.data.frame(Ind_aux1$W2)
  perm2<- perm2 %>% sample_n(nrow(.))
  Ind_aux1$W2<-perm2
  # 2.3.2
```

```
  # Recompute candidate exposure Hfoc for k1 and k2, or H and H2 in the combined
dataset
  Ind_foc1$H<-NA
  Ind_foc1[Ind_foc1$W==1,"H"] <- sum(Ind_foc1$W ==1)+sum(Ind_aux1$W ==1)-1
  Ind_foc1[Ind_foc1$W==0,"H"] <- sum(Ind_foc1$W ==1)+sum(Ind_aux1$W ==1)


  Ind_foc1$H2<-NA
  Ind_foc1[Ind_foc1$W2==1,"H2"] <- sum(Ind_foc1$W2 ==1)+sum(Ind_aux1$W2 ==1)-1
  Ind_foc1[Ind_foc1$W2==0,"H2"] <- sum(Ind_foc1$W2 ==1)+sum(Ind_aux1$W2 ==1)
  # 2.3.3
  Tb_Alg2 = lm(Ydiff ~V1 + V2 + V4 + V5 + V6 + V7 + H + Hdiff, data = Ind_foc1)
# 1.3.3
  summary(Tb_Alg2)
  H_b_Alg2 <- Tb_Alg2$coefficients[8]
  if (Alg2_T0<=H_b_Alg2) # for sum of indicator function notating T0<=Tb
  {
    j=j+1
  }
}
Alg2_P=(1/(B+1))*(1+j)
Alg2_P


# Algorithm 3.
# Using the same 2 experiment data used in algorithm 2
#dset1, dset2
# 3.1.
# Subsetting data from both experiments in to two groups - W=0 and W=1
Ind_0 <- subset(dset1, dset1$W==dset1$W2 & dset1$W==0)
Ind_1 <- subset(dset1, dset1$W==dset1$W2 & dset1$W==1)
# 3.2.
# Arranging data by year as a form of semi-random matching.
Ind_0 <- arrange(Ind_0,Ind_0$V6)
Ind_1 <- arrange(Ind_1,Ind_1$V6)
```

```
Ind_0 <- sample_n(Ind_0,nrow(Ind_1)) # making the data groups same row length

Ind_0l <- Ind_0
Ind_1l <- Ind_1

# 3.3.
# Computing Ydiff_Ind1 and Test statistic
Ind_1$Ydiff_Ind1<-NA
Ind_1$Ydiff_Ind1<- Ind_1$V3-Ind_0$V3

# Computing an alternative test statistic
# Subtracting the same Ydiff as covariates between experiments were unchanged
Alg3_T0 <- mean(Ind_1$Ydiff_Ind1)-mean(Ind_1$Ydiff_Ind1)

# 3.4.
rowsl<-nrow(Ind_1l)
# For loop 3.4.
for (k in 1:B)
{
  # Nested loop for random permutation - binomial randomization with probability
0.5
  for (m in 1:rowsl)
  {
    binom_perm <- rbinom(1, size=1, prob=0.5)
    if (binom_perm==1)
    {
      # Outcomes across experiments are identical in this case,
      # thus permutating only for outcomes of matched groups/indices.
      # In real case scenario when outcomes differ,
      # an identical algorithm can be used for difference between experiments.
      # a1 & b1 as mediator variables
      # to avoid duplication of variables between experiments
```

```
      a1<-Ind_1l$V3[m]
      b1<-Ind_0l$V3[m]
      Ind_1l$V3[m]<-b1
      Ind_0l$V3[m]<-a1
    }}
  # Recomputing Ydiff
  Ind_1l$Ydiff_Ind1<- NA
  Ind_1l$Ydiff_Ind1<- Ind_1l$V3-Ind_0l$V3

  # Recomputing test statistic - difference in means
  Alg3_Tb <- mean(Ind_1l$Ydiff_Ind1)-mean(Ind_1l$Ydiff_Ind1)
  if (Alg3_T0<=Alg3_Tb) # for sum of indicator function notating T0<=Tb
  {
    k=k+1
  }
}
Alg3_P=(1/(B+1))*(1+k)
Alg3_P
```