Designing and Deploying Online Field Experiments

Eytan Bakshy
Facebook
Menlo Park, CA
eytan@fb.com

Dean Eckles
Facebook
Menlo Park, CA
deaneckles@fb.com

Michael S. Bernstein
Stanford University
Palo Alto, CA
msb@cs.stanford.edu

ABSTRACT

Online experiments are widely used to compare specific design alternatives, but they can also be used to produce generalizable knowledge and inform strategic decision making. Doing so often requires sophisticated experimental designs, iterative refinement, and careful logging and analysis. Few tools exist that support these needs. We thus introduce a language for online field experiments called *PlanOut*. PlanOut separates experimental design from application code, allowing the experimenter to concisely describe experimental designs, whether common "A/B tests" and factorial designs, or more complex designs involving conditional logic or multiple experimental units. These latter designs are often useful for understanding causal mechanisms involved in user behaviors. We demonstrate how experiments from the literature can be implemented in PlanOut, and describe two large field experiments conducted on Facebook with PlanOut. For common scenarios in which experiments are run iteratively and in parallel, we introduce a namespaced management system that encourages sound experimental practice.

Categories and Subject Descriptors

H.5.3. [**Group and Organization Interfaces**]: Evaluation / methodology

Keywords

A/B testing; online experiments; toolkits; methodology

1. INTRODUCTION

Randomized field experiments are central to contemporary design and development processes for Internet services. In the most popular case, practitioners use "A/B tests" that randomly assign users to one of two variations of a service. Doing so often allows designers and developers to quickly identify the best choice of the two. The Internet industry has distinct advantages in how organizations can use experiments to make decisions: developers can introduce numerous variations on the service without substantial engineering or distribution costs, and observe how a large random sample of users (rather than a convenience sample) behave when randomly assigned to these variations. So, in many ways, experimentation with Internet services is easy.

For some organizations, randomized experiments play a central role throughout the design and decision-making process. Experiments may be used to explore a design space [19, 20, 33], better attribute outcomes to causes [4, 14, 21], and estimate effects that help decision makers understand how people react to changes and use their services [3, 23, 35]. In this way, even early stages of the user-centered design process can be informed by field experiments. Such trials are not intended to optimize short-term objectives through a pick-the-winner process; they instead aim to provide more lasting, generalizable knowledge. While experiments that achieve these objectives often draw from experimental designs used in the behavioral and social sciences, available tools do little to support the design, deployment, or analysis of these more sophisticated experiments.

Although some experiments are simple to describe, the realities of design, deployment, and analysis, can make experimentation quite complex. Online experimentation is highly iterative, such that preliminary results are used to rapidly run follow-up experiments. This necessitates changing or launching new experiments. Changing live experiments can easily result in statistical inferences that are incorrect. From a development perspective, running follow-up experiments can be time consuming and error prone because experimental logic is often mixed in with application code. Online experimentation is also distributed across individuals and teams, and over time. Distributed work can make it difficult to run experiments simultaneously without interacting with other's experiments or complicating application logic. Combined, these features make it so that experimentation can become so embedded in application code that only a few engineers can correctly modify a particular experiment without introducing errors in its design or future analysis.

In this paper, we discuss how Internet-scale field experiments can be designed and deployed with *PlanOut*, a domain-specific language for experimentation used at Facebook. Designers and engineers working with PlanOut can view any aspect of a service as tunable via parameters: e.g., a flag signaling whether a banner is visible, a variable encoding the number of items in an aggregation, or a string that corresponds to the text of a button. Experimental logic is encapsulated in simple scripts that assign values to parameters. Core random assignment primitives can be combined to reliably implement complex experimental designs, including those that involve assignment of multiple experimental units to multiple factors and treatments with continuous values. PlanOut scripts can also be used as a concise summary of an experiment's design and manipulations, which makes it easier to communicate about, and replicate experiments.

WWW'14, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2744-2/14/04.
http://dx.doi.org/10.1145/2566486.2567967.

PlanOut is enough by itself to author one-off, isolated experiments quickly. However, if experiments must be iterated upon, or related experiments must run in parallel, additional infrastructure is necessary. Such systems can be used to manage experiments and logging, prevent interference between experiments. In this paper, we introduce the architecture of a management system for such situations, and illustrate how iterative experiments can soundly run and analyzed.

In sum, this paper contributes:

- A characterization of online experimentation based on parametrization of user experiences,
- The PlanOut language, which provides a high-ceiling, low-threshold toolkit for parameter-based experiments,
- Guidelines for managing and analyzing iterative and distributed experiments.

These contributions together articulate a perspective on how online field experiments should be conceptualized and implemented. This perspective advocates the use of short, centralized scripts to describe assignment procedures at a high level. It results in experimental practice that is more agile and encourages the production of generalizable, scientific knowledge.

The paper is structured as follows. After reviewing related work in Section 2, we introduce the PlanOut language in Section 3 and show how it can be used to design both simple and complex experiments. Then, in Sections 5 and Section 6, we describe how distributed, iterative experiments can be managed, logged, and analyzed. Finally, we discuss the broader implications and and limitations of our work in Section 7.

2. RELATED WORK

The design and analysis of experiments is a developed area within statistics that is regularly taught, with domain-specific elements, to students in industrial engineering, psychology, marketing, human–computer interaction, and other fields [8, 16, 23, 31]. Tools for producing design matrices for factorial and fractional factorial designs are available (e.g., in the R packages `DoE`, `Design`, and `rms`). Such packages are useful for designing small scale studies, but since the design matrix is created *a priori*, they are not well suited for online settings where newly created experimental units must be assigned in real-time and assignment may depend on unit characteristics not available in advance.

The prevalence of randomized experiments in the Internet industry and the tools developed there are only partially represented in the scholarly literature. Mao et al. [24] created experimental frameworks for crowdsourcing sites such as Amazon Mechanical Turk. Several papers by Kohavi et al. present recommendations on how to implement and instrument experiments [19], as well as common pitfalls in analyzing experiments [12, 18].

Existing experimentation tools include associating experiments with “layers” (at Google and Microsoft [18, 32]) or “universes” (at Facebook), such that all conditions in the same layer are mutually exclusive. Some tools (e.g., Google Analytics, Adobe Target) include mechanisms for associating condition identifiers with configuration information (e.g., dictionaries of parameters and their values).

While the types of experiments we focus on in this paper are designed to inform product decision-making, other experiments are run simply to optimize a single outcome variable. For example, another active area of development focuses on implementing heuristics for optimizing stochastic functions (e.g., multi-armed bandit optimization) [22, 30].

3. THE PLANOUT LANGUAGE

The PlanOut language separates experimental design from application logic and focuses experimenters on the core aspect of an experiment: how *units* (e.g., users, items, cookies) are randomly assigned to conditions, as defined by parameters (e.g., settings for user interface elements, references to ranking algorithms). PlanOut promotes a mental model where every aspect of the site is parameterizable, and experiments are a way of evaluating user experiences defined by those parameters.

This approach encourages experimenters to decompose large changes into smaller components that can be manipulated independently. In doing so, experimenters are better equipped to attribute changes in user behavior to specific features and thus inform design decisions. Decomposition also allows experimenters to more easily iterate on experiments so that some features remain fixed while others change.

Experimenters use PlanOut by writing a PlanOut script, which may then be executed via an API for each unit (e.g., user or user-story combination). Each script indicates which inputs are used for assignment, how random assignment should occur, and the names of parameters that can be accessed via the API and logged. PlanOut scripts are executed sequentially. In later sections, we introduce a number of scripts for both standard and complex experiments.

From a systems perspective, PlanOut is a way of serializing experiment definitions (e.g., as JSON) so they can be easily stored and executed on multiple platforms, such as layers of backend and frontend services, and mobile devices. Serialized PlanOut code (Figure 1c) can be generated and edited through a domain-specific language (DSL) (Figure 1a) or graphical user interfaces (Figure 1b). The DSL is presented in the remaining sections, but many of these examples could alternatively be formulated through the use of graphical interfaces.

The PlanOut DSL and its syntax are minimal. The primary contribution of the language is in providing a parsimonious set of operations for thinking about, designing, and implementing experiments. Because of this, we spend little time discussing the language itself or its built-in operators. A complete list of operators can be found in the documentation for the reference implementation of PlanOut.[1]

3.1 Functionality

We begin our discussion of PlanOut by giving several illustrative examples of how scripts and operators work. We first describe a simple A/B test and show how it can be generalized into factorial designs. Then we consider how experimental designs that involve multiple types of units in the randomization of a user interface element can be used to estimate different effects. Finally, we discuss conditional evaluation (e.g., for pre-stratification), and other operators.

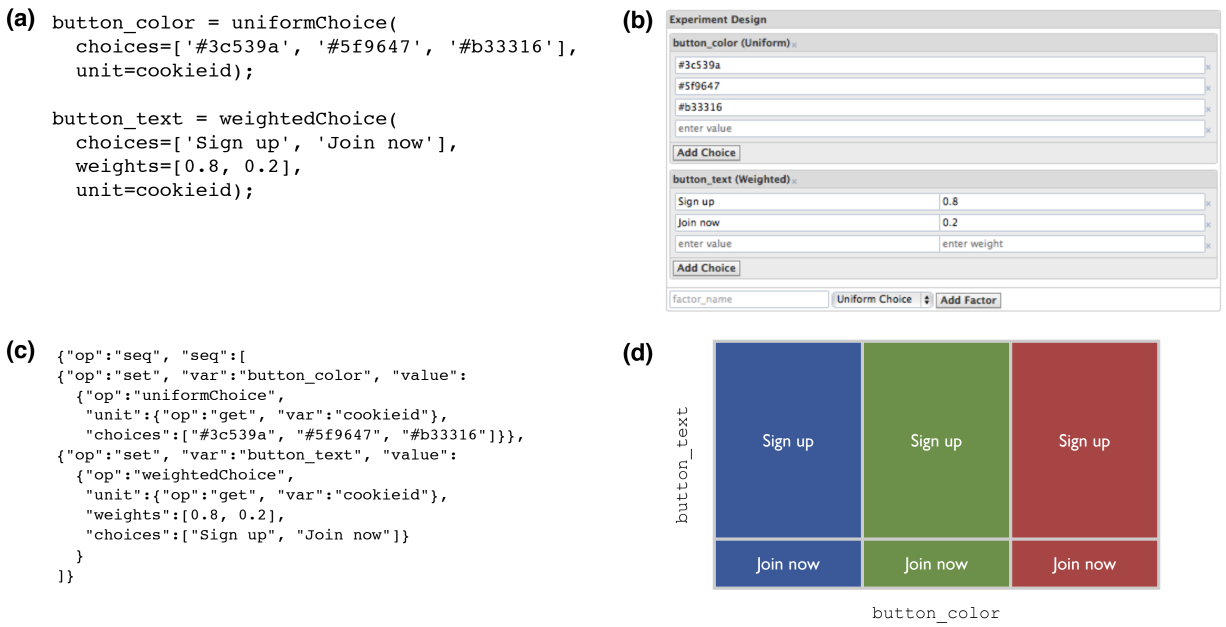
3.1.1 A/B test

The most common type of experiment involves uniform random selection — for example randomly assigning users to one of several button colors or text options. This can be accomplished via the `uniformChoice` operator,

```
button_color = uniformChoice(
  choices=['#3c539a', '#5f9647', '#b33316'],
  unit=cookieid);
```

Here, each `cookieid` is assigned deterministically to one of three

[1] An open source implementation of a PlanOut interpreter and API is available at `https://github.com/facebook/planout`.



(a)

```
button_color = uniformChoice(
  choices=['#3c539a', '#5f9647', '#b33316'],
  unit=cookieid);

button_text = weightedChoice(
  choices=['Sign up', 'Join now'],
  weights=[0.8, 0.2],
  unit=cookieid);
```

(b)

Experiment Design

button_color (Uniform)

#3c539a	
#5f9647	
#b33316	
enter value	

Add Choice

button_text (Weighted)

Sign up	0.8
Join now	0.2
enter value	enter weight

Add Choice

factor_name | Uniform Choice | Add Factor

(c)

```
{"op":"seq", "seq":[
{"op":"set", "var":"button_color", "value":
  {"op":"uniformChoice",
   "unit":{"op":"get", "var":"cookieid"},
   "choices":["#3c539a", "#5f9647", "#b33316"]}},
{"op":"set", "var":"button_text", "value":
  {"op":"weightedChoice",
   "unit":{"op":"get", "var":"cookieid"},
   "weights":[0.8, 0.2],
   "choices":["Sign up", "Join now"]}
  }
]}
```

(d)

Figure 1: A factorial experiment in PlanOut. (a) PlanOut language script (b) a graphical interface for specifying simple PlanOut experiments (c) a JSON representation of serialized PlanOut code (d) an illustration of the proportion of `cookieids` allocated to each parameterization. Note that because we use `weightedChoice()` to assign `button_text`, more cookies are assigned to "Sign up" than ''Join Now".

possible button colors. In application code, the experimenter will later be able to evaluate this PlanOut script for a particular `cookieid` (e.g., in the case of a Web-based sign-up form), and retrieve the runtime value through the variable name `button_color`.

3.1.2 *Multifactor experiment*

Creating a full factorial experiment means setting multiple variables that are evaluated independently. For example, suppose we wanted to manipulate not just the color of a button but also its text. The script for this experiment is given in Figure 1a, and includes two operators, a `uniformChoice` and a `weightedChoice`. We use `weightedChoice` to assign (on average) 80% of the cookies to have the button text "Sign up", and 20% to have the text "Join now". Setting these two parameters as shown in Figure 1 generates $2 \times 3 = 6$ conditions, whose proportions are summarized in Figure 1d.

3.1.3 *Conditional execution*

Many experiments cannot be described through fully factorial designs, e.g., in the case where some values of one parameter may only be valid when another parameter is set to a particular value, or assignment probabilities are dependent on another variable. PlanOut thus includes operators for conditional control flow, such as `if` / `else`, boolean operations (i.e., `and`, `or`, `not`), comparison operations (e.g., `==`, `>=`) and array indexing.

Consider a scenario where we wish to control the population of users receiving a new translation feature so that a higher proportion of US users receive the feature. To accomplish this, one could pass in both a `userid` and `country` to the PlanOut interpreter, and use conditional logic,

```
if (country == ‘US’) {
  has_translate = bernoulliTrial(p=0.2, unit=userid);
} else {
  has_translate = bernoulliTrial(p=0.05, unit=userid);
}
```

or alternatively, via array indexing,

```
strata_p = [0.05, 0.2];
has_translate = bernoulliTrial(
  p=strata_p[country == ‘US’],
  unit=userid);
```

Here, arrays are zero-indexed and true/false evaluate to `1` and `0`, respectively.

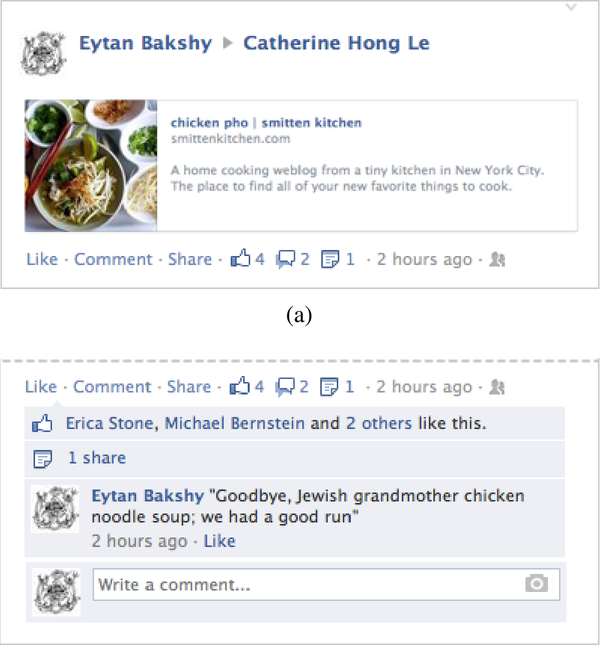
3.1.4 *Experiments with multiple and nested units*

Many effects are better understood through randomization of units other than the user. For instance, while most standard A/B tests are between-subjects designs, where users are randomly assigned to different experiences, some effects may be more precisely estimated through a within-subjects design. These experiments can be implemented by transitioning from a single experimental unit (e.g., `viewerid`) to tuples of units.

Consider an experiment that manipulates whether a story in users' News Feed has its comment box collapsed (Figure 2a) or expanded (Figure 2b). If an experimenter wanted to assign 5% of all News Feed items to have a collapsed comment box, so that users must click to see comments attached to a story, one could define such an experiment by:

```
collapse_story = bernoulliTrial(p=0.05,
  unit=[viewerid, storyid]);
```

The `bernoulliTrial` operator returns `1` with probability `p`, and `0` otherwise. By making `unit` a tuple, `[viewerid, storyid]`, one achieves a fully randomized within-subjects design, where each user sees, in expectation, an independent 5% of posts with collapsed comment boxes. This type of design may be used to estimate the effect of collapsing individual stories on individual viewers' responses to that story (e.g., likes, comments).



Eytan Bakshy ▸ Catherine Hong Le

chicken pho | smitten kitchen
smittenkitchen.com

A home cooking weblog from a tiny kitchen in New York City. The place to find all of your new favorite things to cook.

Like · Comment · Share · 4 2 1 · 2 hours ago ·

(a)

Like · Comment · Share · 4 2 1 · 2 hours ago ·

Erica Stone, Michael Bernstein and 2 others like this.

1 share

Eytan Bakshy "Goodbye, Jewish grandmother chicken noodle soup; we had a good run"
2 hours ago · Like

(b)

Figure 2: (a) A News Feed story whose comment box is collapsed and (b) an expanded comment box. Different causal effects can be estimated by randomizing the state of the comment box over different experimental units (e.g., source users, viewers, stories).

Other ways of assigning units can be used to estimate different quantities. If one were to instead assign `viewerids` to conditions, 5% of users would see all stories collapsed, which could have large effects on how viewers interact with all stories and how many stories they would consume. Had `storyid` been the randomized unit, particular stories would be collapsed or not collapsed for all viewers. Because viewers are expected to comment at a higher rate when the box is expanded, randomizing over just `storyid` can produce herding effects. These differences result from interference or "spillovers" across conditions [1, 29], and highlights how supporting multiple experimental units can be useful for evaluating how user interface elements affect complex user dynamics.

3.1.5 Extensions

PlanOut is extensible. If an assignment procedure cannot be implemented using built-in operators, developers may write custom operators in a native language (e.g., PHP or Python), including those that integrate with other services. PlanOut experiments at Facebook often use custom operators that interface with gating infrastructure, graph cluster randomization [34], and other experimentation systems. Classes for random assignment are also extensible, so that procedures can be easily implemented using the hashing methods described below.

3.2 Random assignment implementation

Many operators involve generating pseudo-random numbers deterministically based on input data (e.g., a user generally should be assigned to a the same button color each time they load a particular page). A sound assignment procedure maps experimental units to parameters in a way that is deterministic, as good as random, and unless by design, independent of other parameter values from the same or other experiments. And because experiments can be linked across multiple service layers (e.g., ranking *and* user interfaces), it is important that any pseudo-random operation can be kept consistent across loosely-coupled services that may be written in different languages.

The PlanOut interpreter implements procedures that automatically fulfill these requirements. Rather than using a pseudo-random number generator, or directly hashing units into numbers, the interpreter "salts" inputs to a hash function so that each assignment (unless otherwise specified) is independent of other assignments, both within and across experiments. Because the procedure based off of standard hashing functions (i.e., SHA1), it is deterministic and platform-independent. At a low level, this is done by prepending each unit with a unique experiment identifier and variable-specific salt. Thus, the hash used to assign a variable such as `button_color` is not just the input ID (e.g., `42`), but instead, e.g., `user_signup.my_exp.button_color.42`, where `user_signup` is the namespace of the experiment and `my_exp` is the identifier of the particular experiment (namespaces and experiments are more specifically defined in following sections).

4. EXAMPLE EXPERIMENTS

The preceding examples show how PlanOut is a low-threshold language for implementing basic experiments. In this section we demonstrate that PlanOut also has a high ceiling, in that complex, scientific experiments can be concisely specified in only a few lines of code.

4.1 Examples from prior research

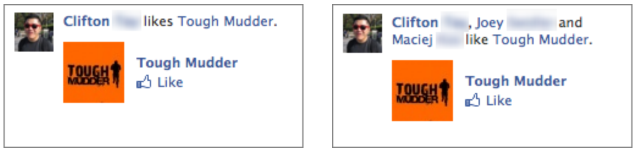
We begin by demonstrating how one could implement two published experiments from the social computing literature.

4.1.1 Experimenting with goal-setting

In an influential application of social psychological theory to online systems, Beenen et al. [6] experimented with strategies for encouraging users to contribute ratings to the movie recommendation service and online community MovieLens. In Study 2, they randomly assigned users to an email that sets a goal for users to rate movies. Users were either identified as being part of a group, and having a group-level goal, or having an individual goal. The goal was either specific or a "do your best" goal; if specific, it was a number of movies to be rated in a week scaled by the size of the group. This experiment could be implemented as:

```
group_size = uniformChoice(choices=[1, 10],
  unit=userid);
specific_goal = bernoulliTrial(p=0.8, unit=userid);
if (specific_goal) {
  ratings_per_user_goal = uniformChoice(
    choices=[8, 16, 32, 64], unit=userid);
  ratings_goal = group_size * ratings_per_user_goal;
}
```

This experiment could then be analyzed in terms of the per-person specific goal, as in Beenen et al. [6]. There are multiple other ways to implement this experiment, some of which correspond to different choices about how to split logic between PlanOut and the application code. For example, the actual text used in the emails could be constructed in the PlanOut code, while the implementation above follows from the judgement that it would be better to do so in the application logic.Such choices can also depend on other available tools, such as tools for automatically creating translation tasks for new strings used in an online service.



Clifton [redacted] likes Tough Mudder.

Tough Mudder

Like

Clifton [redacted], Joey [redacted] and Maciej [redacted] like Tough Mudder.

Tough Mudder

Like

(a) (b)

Figure 3: A within-subjects experimental design that deterministically randomizes social cues presented to users. Example from Bakshy et al. [3] when (a) 1 of 3 and (b) 3 of 3 cues are shown.

4.1.2 *A social cues experiment with complex inputs*

Consider an experiment on the effects of placing social cues alongside advertisements from Bakshy et al. [3]. A small percentage of user segments were allocated to this experiment (see Section 5.1); for these users, some social cues were removed from ads. For instance, if a user in the experiment had three friends that "like" a particular Facebook page being advertised, then this user would be randomly assigned to see one, two, or three friends associated with the page (Figure 3). This experiment can be written as follows:

```
num_cues = randomInteger(
  min=1, max=min(length(liking_friends), 3),
  unit=[userid, pageid]);
friends_shown = sample(
  choices=liking_friends, draws=num_cues,
  unit=[userid, pageid]);
```

The input data to the PlanOut experiment would be `userid`, `pageid`, and `liking_friends`, an array of friends associated with the page. The script specifies that each user–page pair is randomly assigned to some number of cues, `num_cues`, between one and the maximum number of displayable cues (i.e., no more than three, but no greater than the number of friends eligible to be displayed alongside the ad). That number is then used to randomly sample `num_cues` draws from `liking_friends`. That is, the script determines both the number of cues to display and the specific array of friends to display.

4.2 Experiments deployed using PlanOut

The following represent a sample of experiments that have been designed and deployed using PlanOut at Facebook.

4.2.1 *Voter turnout experiment*

In an extension and replication of prior experiments with voter turnout [7, 9], an experiment assigned all voting-aged US Facebook users to encouragements to vote in the 2012 Presidential Election. This experiment involved assigning users to both a banner at the top of the screen and eligibility for seeing social stories about friends' self-reported voting behavior in News Feed. We show a subset of the parameters set by this experiment.

```
has_banner = bernoulliTrial(p=0.97, unit=userid);
cond_probs = [0.5, 0.98];
has_feed_stories = bernoulliTrial(
  p=cond_probs[has_banner],
  unit=userid);
button_text = uniformChoice(
  choices=["I'm a voter", "I'm voting"],
  unit=userid);
```

We can see that `has_banner` is `1` for 97% of users. Then, we define `cond_probs` to be the conditional probability that one would show feed stories given that `has_banner` is either `0` or `1`. We assign `has_feed_stories` using a `bernoulliTrial()` with `p=cond_probs[has_banner]`, so that those with the banner have a high chance of also being able to see the feed stories, and those without a banner have an equal probability of being able to see or not see the feed stories. Finally, the button text for the call to action in the banner is subject to experimental manipulation (provided that the user is in the `has_banner=1` condition). Analyses can then examine effects of the banner, effects of social stories about voting, and its interaction with verb/noun phrasing [9].

4.2.2 *Continuous-treatment encouragement design*

Encouragement designs [17] randomize an inducement to a behavior of interest so as to evaluate the inducement or study the behavior's downstream effects. In online services, it is common to encourage users to engage with a particular entity, user, or piece of content. For instance, if having more ties on Facebook is hypothesized to increase long-term engagement, one could establish a causal relationship by randomizing whether or not some users receive recommendations for additional friends.

Evidence suggests that users who receive more feedback on Facebook are more likely to become engaged with the site [10]. If there is a (forward) causal relationship between these variables, then changes to the site that affect how much feedback users receive can in turn affect user engagement and content production.

The following experiment examines this hypothesized effect by randomizing encouragements for friends to engage with a source user's content. It also illustrates random assignment involving multiple units. As mentioned in Section 3.1.4, expanding or collapsing News Feed stories can affect engagement with stories. The script below randomly assigns each source user to a proportion, such that on average, that proportion of the source's friends see a collapsed comment box when stories they produce appear in News Feed.

```
prob_collapse = randomFloat(min=0.0, max=1.0,
  unit=sourceid);
collapse = bernoulliTrial(p=prob_collapse,
  unit=[storyid, viewerid]);
```

Each source user is assigned to a probability `prob_collapse` in $[0, 1]$. Then, each story–viewer pair is assigned to have a collapsed comment box with probability `prob_collapse`. To carry out this assignment, we invoke the PlanOut script from the part of News Feed rendering logic that determines whether stories' comment boxes should be expanded or collapsed, using `sourceid`, `storyid`, and `viewerid` as inputs (more discussion of the application interface is covered in Section 5.3.1).

There a number of possible ways this experiment can be analyzed. First, one can identify the effect of modulating feedback encouragements on the total amount of feedback a user's stories get. Second, we can test our original hypothesis that feedback causes users to engage more with the site (e.g., log in more often or produce more content). One can look at the effect of the assignment to different values of `prob_collapse` on users' engagement levels, or use an instrumental variables analysis [17, 27], which combines estimates of effects of the encouragement on feedback received and engagement, to estimate the effect of feedback on users' engagement.

5. RUNNING EXPERIMENTS

We have discussed ways of designing and executing randomized assignment procedures, but have not described how experiments are managed, tested, and logged. Here we define an *experiment* to refer to a PlanOut script combined with a target population for which that script was launched at a specific point in time. From the perspective of the experimenter and logging infrastructure, different experiments are considered separately.

In the following subsections, we describe a broader technical context for running experiments. This supporting infrastructure includes: a system for managing and logging experiments, including a segmentation layer which maps units to experiments, a launch system which provides default values for parameters not subject to experimental manipulation, an API for retrieving parameters, and a logging architecture which simplifies data analysis tasks.

5.1 Namespace model of site parameters

Field experiments with Internet services frequently involve the manipulation of persistent parameters that are the subject of multiple experiments, whether conducted serially or in parallel. We use a namespace model of parameters to support these practices.

Experimentation is frequently iterative; as in scientific research, a single experiment is often not definitive and so requires follow-up experiments that manipulate the same parameters. A second experiment with a near-identical design may be used to more precisely estimate effects, or might include new variations suggested by the first's results or other design work. Continual redesigns and development might also change the effects of the parameters, thus motivating the need for additional experiments.

Likewise, multiple experiments manipulating the same aspects of a service are frequently run in parallel, sometimes by different teams with minimal explicit coordination. Two teams may manipulate (a) the same features of the same service, (b) independent features of the same service (e.g., font size and items per page), or (c) different layers of the same service (e.g., link colors and the ranking model which selects which items are to be displayed). In these cases, it is helpful to have an experimentation system that can keep track of and/or restrict which parameters are set by each experiment. This requires that experimentation tools be cross-platform and can handle allocation of units to multiple experiments started at different times by different teams.

The solution to support these practices within the PlanOut framework is to use *parameter namespaces* (or namespaces for short). This is a natural extension of thinking of the experimentation system as being how many parameter values are read in application code; often these parameters are an enduring part of the service, such that, over time, many experiments will set a particular parameter. Each namespace is centered around on a primary unit (e.g., users). A new experiment is created within a new or existing namespace by allocating some portion of the population to a PlanOut script.[2]

5.2 Experiment management

Experiments can be managed as follows: for each namespace, hash each primary unit to one of a large number (e.g., 10,000) of *segments*, and then allocate individual segments to experiments. This segment-to-experiment mapping may be maintained in a database or other data storage system. Similar to segmentation systems discussed in prior work [32], when a new experiment is created, it is allocated a random set of segments. These segments are deallocated once the experiment is complete.

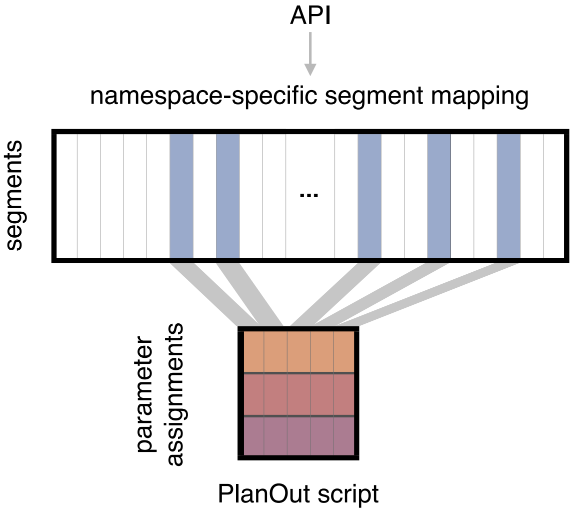


Figure 4: Blocks of segments (e.g., multiple buckets of user ids) are assigned to experiments, which map experimental units to parameters in a way that is uncorrelated with the segmentation.

Each experiment's script makes no reference to these segments, such that random assignment to parameter values within each experiment is independent of the segmentation (Figure 4).[3] This feature is accomplished via the hashing method described in Section 3.2, and reduces the risk of carryover effects [8] that might occur if whole segments from one experiment were all assigned to the same parameterizations and subsequently reallocated to a different experiment [18, 19].

Namespaces can be represented in experimentation management tools as a dashboard of currently running experiments along with a listing of parameters for that namespace. When an experiment is complete, its segments can become available for future experiments. When iteratively experimenting, new versions of an experiment can be created and allocated new segments. For example, when experimenters need to increase precision by experimenting with more units, engineers can duplicate the experiment definition and allocate additional segments to the experiment. Increases in size are often coupled with changes to the experiment definition, e.g., adding new parameterizations similar to promising versions. Such iteration is generally preferable to modifying the existing experiment, which can frequently produce problems in analysis (see Section 6).

5.2.1 Parameter defaults

In some cases, all units will have a parameter set by an experiment. For example, an experimenter working with a new parameter in a new namespace may simply allocate all segments to an experiment. But in other cases, some units will not have a particular parameter set by any experiment. Then the value of the parameter used for those units can be set in some other way. Often, this would reflect the status quo and/or what values are currently believed to

[2] Any two experiments which set the same parameter must be mutually exclusive of one another (i.e., must assign parameter values only for disjoint sets of units). More generally, consider the graph of experiments in which two experiments are neighbors if they set any of the same parameters. It is then natural to require that all experiments in the same connected component to be mutually exclusive. This motivates the idea of using namespaces to group parameters that are expected to be set by experiments in the same connected component of this graph.

[3] This corresponds to what Kohavi et al. [18] call "local randomization". It requires that each experiment have its own dedicated control group, which they identify as the approach's lone disadvantage.

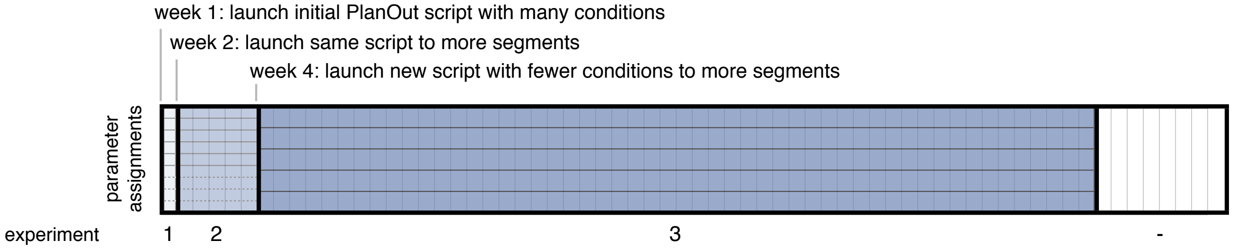

week 1: launch initial PlanOut script with many conditions
week 2: launch same script to more segments
week 4: launch new script with fewer conditions to more segments
parameter assignments
experiment
1
2
3
-

Figure 5: An illustration of how namespaces are used to launch experiments. Segments (vertical regions) are randomly allocated to experiments; here we order the segments by which experiment they were allocated to. Segments not allocated to experiments use the default parameter values. Results from each experiment are generally analyzed separately. Horizontal lines distinguish conditions in each experiment; dotted lines indicate the conditions removed in experiment 3.

be optimal. This same value is likely assigned by other experiments (e.g., if they include "control" conditions).[4]

More technically, if a request for a parameter is made but that parameter is not set by an experiment (i.e., the unit is not assigned to an active experiment, or the unit is assigned to an active experiment that does not set the parameter), then some specified *launch value* is used. If this launch value is not specified, or the experimentation service fails, the default specified in the application code is used instead. Other extensions, such as having launch values vary depending on unit characteristics (e.g., country) may also be implemented in a straightforward way.

5.2.2 *Workflow for iterative experimentation*

We summarize how the tools presented here fit together by describing an iterative experiment (Figure 5). First, we created a namespace for a particular user interface, implemented front-end code to retrieve parameter values from that namespace immediately before rendering UI elements, and set the default launch values to settings that had previously been hard-coded in PHP. In our case, logging for the outcomes of interest was already instrumented, so no additional instrumentation was needed.

Then, we scripted an initial PlanOut-based experiment and launched it to a small set of users (experiment 1 in Figure 5). After one week of observing that the experiment did not cause statistically significant decreases in key metrics, this same script was launched to roughly 8 times the number of users (experiment 2). Results were initially analyzed using internal tools, and then in greater detail using `R`. Because the experiment was expected to have long-term effects, results from each experiment were analyzed separately even though they used the same script. We found that the primary outcomes were clearly worse under parameterizations involving a particular parameter value, and that higher values of a second parameter appeared to increase one outcome at the expense of another outcome.

Based on these results, we created a new PlanOut script that did not include clearly suboptimal parameterizations, and extended the range of the parameter we hypothesized to represent an important tradeoff. This new script was launched in a third experiment to an additional segment of users (experiment 3 in Figure 5), and analyzed for several weeks.

After considering longer-term results from the three experiments, we decided on a parameterization to use as a default for all users. We de-allocated all segments, set a new default parameterization, and created a new, smaller experiment that assigned users to the new and original parameterizations with equal probability. This fourth experiment, commonly referred to as a "backtest", is used to evaluate the efficacy of the launch decision after a long period of time.

While this type of backtest is easy to implement and avoids the potential for downstream errors in analysis, there are a number of possible ways we could have run the backtest. If a prior experiment (e.g., experiment 3) had reasonable power for the comparison between the old and new defaults, we could continue to run that experiment and disable all other parameterizations. Users assigned to disabled parameterizations would take on the new default parameterization, but would not be used in the subsequent analysis of the experiment. This approach can be particularly attractive if the experiment is expected to have time-varying effects, or if one wanted to minimize changes to individual users' experiences. A third option would involve running a new experiment in the unallocated segments, if there are enough such segments after experiments 1–3.

5.3 Integration with application code

5.3.1 *Application programming interface*

At runtime, application code needs to retrieve parameter values within a namespace for particular units. This can be done by constructing an object which interfaces with management and logging components. For example, retrieving the parameter `collapse_story` associated with a particular viewer-story pair in the `comment_box` namespace might be invoked by instantiating an experiment object for the input units, and request the needed parameters:

```
exp = getExp('comment_box',
  {'viewerid':  vid, 'storyid':  sty})
collapse_story = exp.get('collapse_story')
```

The management system would then map the units to a segment within the `comment_box` namespace, which gets mapped to an experiment and its respective PlanOut script. The script is executed with the input data, and if the script that sets a parameter requested by `get()`, the value is returned and the event is logged. If the requested parameter is not set, then the parameter default (described in Section 5.2.1) is used. Because assignment procedures bear some computational cost, and are generally deterministic, parameter assignments can be cached.

[4] From a statistical perspective, it may seem that not counting these units as part of a control group makes for inefficient analysis. This is sometimes true, but doing so would place additional requirements for standard analyses to be correct (e.g., requirements on the history of values assigned to units not in experiments). When these requirements are satisfied, an analyst could make use of that data as needed.

5.3.2 Testing experiments and parameter values

When designing experiments, engineers often need to be able to test a service under a range of parameter values. While PlanOut scripts can be difficult to interact with directly (as they are not written in a native language, like PHP), they can still be tested and debugged in situ with a small amount of additional infrastructure. In particular, by providing developers with a way to override or "freeze" parameters so that they maintain a prespecified value throughout a PlanOut script's execution, one can test assignment to conditions even if few units (or combinations of units) are assigned to them, without modifying any application or PlanOut code.

This functionality can be surfaced to Web developers via URL query parameters. Freezing the `has_feed_stories` parameter to `1` in the voter turnout experiment described in Section 4.2.1 (running within the `vote2012` namespace) may then be accomplished by accessing a URL like:

```
http://...php?ns_vote2012=has_feed_stories:1
```

Freezing also allows one to test downstream effects of different inputs. Overriding `userid` or `has_bannner` may in turn change whether feed stories are shown. Combinations of parameters can also be frozen by specifying a list of parameters to be set, e.g. `has_banner:1,has_feed_stories:0`. Overrides for mobile applications or backend services may alternatively be set through server-side management tools.

5.4 Logging

Logging occurs automatically when `get()` is called, so that there is a record of the exposure of units to the experiment. By default, the namespace, the experiment name, all input data, and variables set by the PlanOut script are logged. This type of *exposure logging* has a number of benefits, including simplifying downstream data analysis and increasing statistical power by distinguishing between units that may have been affected by assignment and those that are known to be unaffected. (For example, many users who are assigned to be in an experiment may not actually arrive at the part of the site that triggers the manipulation, and thus their outcome data should not be included in analysis in the normal way.) As with experimental assignment, caching of prior exposures help reduce load on experimental infrastructure. It is also sometimes desirable to log auxiliary information not related to the assignment, including user characteristics or events (i.e., "conversions"). This might be done through a separate method (e.g., `log()`) in the experiment object. Exposure logging, combined with the management system, prevents a number of common pitfalls we have observed at Facebook. These benefits are discussed in the following section.

6. ANALYZING EXPERIMENTS

While domain-specific aims and especially complex experimental designs will often require custom analysis, much analysis of online experiments can be automated in support of their routine and valid use in decision making. This eliminates common sources of errors in analysis, makes results more directly comparable, and reduces the burden of running additional experiments. This kind of automation is easy to accomplish with PlanOut experiments because their scripts directly encode a representation of their parameters, values, and design. A complete description of accompanying systems for analyzing experiments is beyond the scope of this paper, but we discuss how the design of PlanOut interacts with common analyses, automated or not.

Logging only users who have received the treatment (versus analyzing the entire population who could could have potentially be exposed) improves statistical inference in two ways. First, it can substantially decrease the variance of estimated treatment effects when the number of users exposed to an experimental manipulation is small relative to the number who are assigned (e.g., in the case of a less commonly used or new feature). This reduces risk of Type II errors, in which an experiment has an effect, but experimenters are unable to detect that effect. Secondly, exposure logging focuses experimenters on a more relevant sub-population whose outcomes are plausibly affected by the treatment.

Explicit logging of labeled input units and assigned parameter values affords flexibility in terms of automated analysis. Many relatively simple designs can be fruitfully analyzed by computing summary statistics for outcomes of interest for each unique combination of parameter values. Since parameters are logged for each exposure, analyses of full or fractional factorial designs that make use of this structure can also be automated. For example, questions about main effects of factors (e.g., "does button color have any average effect?") can be answered via an analysis of variance. Representation of the experiment in terms of parameters can also make it easier to automatically use this structure in estimating expected outcomes for each condition. For example, systems can fit penalized regression models with the main effects and all relevant interactions, thus "borrowing strength" across conditions that have the same values for some parameters [15]. These types of model-based inference also help reduce the risk of not being able to detect clinically significant changes. We have found that when these forms of analyses are not possible, engineers and decision makers tend to avoid more complex experimental designs, like factorial designs, because they tend to be underpowered, even though they have a number of benefits for improving understanding and identifying optimal treatments.

6.1 Analyzing iterative experiments

Iteration on experiments with PlanOut occurs primarily through creating new experiments that are variants of previous experiments. By default, these experiments are then analyzed separately, which avoids several problems that can occur when attempting to pool data from before and after a change in experimental design. For example, adding additional users to an existing experiment but assigning them to new conditions means that these users are first exposed to their treatment more recently than other users; comparisons with other conditions can be biased by, e.g., novelty effects or cumulative treatment effects.

There are some cases where two similar experiments can be analyzed together. For example, if two experiments' PlanOut scripts are identical but have different numbers of segments allocated to them and are started at different times, they can be pooled together to increase power, though this changes what is estimated to a weighted average of potentially time-varying effects. This may result in underestimation or overestimation of treatment effects, and highlights the ways in which gradual product rollouts might be better represented as experiments. More sophisticated automatic selection of analyses that pool data across experiments remains an area for future work.

6.2 Units of analysis

Many online experiments use a small number of standard types of units, for which outcomes of interest may already be available in data repositories. For example, most experiments at Facebook involve random assignment of user IDs, and the standard desired analysis involves analysis of behaviors associated with user IDs. Other cases can be more complex. For example, an experiment may randomly assign users and advertisements, userid–advertisementid pairs, to parameter values, but it may be necessary for an analysis to account for dependence in multiple observations of the same user or ad to obtain correct confidence intervals and hypothesis tests [2, 11]. Inspection of a script can identify the units for which different parameters are randomized, which can be used in subsequent selection of methods for statistical inference.

7. DISCUSSION

Randomized field experiments are a widely used tool in the Internet industry to inform decision-making about product design, user interfaces, ranking and recommendation systems, performance engineering, and more. Effective use of these experiments for understanding user behavior and choosing among product designs can be aided by new experimentation tools. We developed PlanOut to support such scalable, randomized parameterization of the user experience. Our goal has been to motivate the design of PlanOut using our experiences as experimenters and by demonstrating its ability to specify experimental designs, both simple and complex, from our work and the literature.

One aim of conceptualizing experiments in terms of parameters and enabling more complex experimental designs is that online experiments can be more effectively used for understanding causal mechanisms and investigating general design principles, rather than simply choosing among already built alternatives. That is, PlanOut aims to support uses of randomized experiments more familiar in the sciences than in the Internet industry. For example, the primary purpose of the social cues experiment described in Section 4.1.2 is not to decide whether it is better to show fewer social cues alongside ads (doing so was expected to and did reduce desired behaviors), but to estimate quantities that are useful for understanding an existing service, allocating design and engineering resources, and anticipating effects of future changes.

In addition to being a means for deploying Internet-scale field experiments, we have found PlanOut to be useful aid for describing and collaborating on the design of complex experiments, well before they are deployed. We hope others will also find the notation to be a clear way describe their experiments, whether in face-to-face settings or documentation of published research. The PlanOut language itself may also be applicable to other types of experiments, such as those conducted on Amazon Mechanical Turk [25].

There are important limitations to online experiments in general and PlanOut in particular. As others have argued, randomized experiments cannot effectively replace all other methods for learning from current and potential users [28] and anticipating effects of future interventions [13]. Most notably, random assignment of users to a new version of a service requires that that version is built and of sufficient quality. Nielson [28] additionally argues that A/B tests encourage short-term focus and do not lead to behavioral insights. While this is perhaps a fair critique of many widespread experimentation practices, PlanOut is designed to run experiments that lead to behavioral insights, modeling, and long-term learning.

Even though field experiments are the gold standard for causal inference, their results are also subject to limitations. Because the underlying effects can be heterogeneous and dynamic, results of a field experiment from one time and one population may not generalize to new times and populations [5, 23, 35]. One hope that we have for PlanOut is that it encourages more sophisticated behavioral experiments that allow estimation of parameters that are more likely to generalize to future interfaces. PlanOut and the associated infrastructure also make it easy to replicate prior experiments. Finally, standard experimental designs and analyses do not account for one unit's outcomes being affected by the assignment of other units (e.g., because of for peer influence and network effects) [1, 29]. In the presence of such interference, user behavior can substantially change post-launch behaviors as connected users interact with one another [34].

PlanOut has more specific limitations with respect to designs where one unit's assignment depends on the assignment of a large number of other units. Random assignment schemes that involve optimizing global characteristics of the experimental design are thus more difficult to implement directly using built-in operators. This includes pre-stratified or block-randomized designs [8, 16] that use sampling without replacement in a prior, offline assignment procedure, but in online experiments these designs usually offer minimal precision gains.[5] Assignment schemes such as graph cluster randomization [34], which involves partitioning the social network of users, require offline computation. In such cases, PlanOut may simply be a useful framework for providing a consistent interface to accessing information about units (e.g., the results of the graph partitioning) that has been computed offline through a custom operator and then assigning parameter values based on that information. Sequential experimentation techniques, such as multi-armed bandit heuristics [30] are another such example where custom operators are generally needed.

We have only briefly discussed how online experiments should be analyzed. From our experience, the availability of easy-to-use tools for routine analysis of experiments has been a major factor in the adoption of randomized experiments across product groups at Facebook; so the automation of analysis deserves further attention. This also suggests another area for future research: How should we best evaluate new tools for designing, running, and analyzing experiments? We have primarily done so by appealing to prior work, our own professional experiences, and by demonstrating the expressiveness of the PlanOut language. Deciding whether an experiment was "successful" or effective can depend on broader organizational context and hard-to-trace consequences as an experiment's results diffuse throughout an organization. Some of the most effective experiments directly inform decisions to set the parameters they manipulate, but other well-designed experiments can be effective through broader, longer-term influences on beliefs of designers, developers, scientists, and managers.

8. ACKNOWLEDGEMENTS

As described here, PlanOut is only a small piece of the broader set of experimentation tools created by our colleagues. At Facebook, PlanOut runs as part of QuickExperiment, a framework developed by Breno Roberto and Wesley May. The perspective we take on how experiments should be logged and managed is greatly influenced by previous tools at Facebook and conversations with Daniel Ting, Wojciech Galuba, and Wesley May. The design of PlanOut was influenced by conversations with John Fremlin, Brian Karrer, Cameron Marlow, Itamar Rosenn, and those already mentioned. Finally, we would like to thank Brian Davison, René Kizilcec, Winter Mason, Solomon Messing, Daniel Ting, and John Myles White for their comments on this paper. René Kizilcec built the PlanOut GUI in Figure 1(c). John Fremlin built the PlanOut DSL to JSON compiler.

9. REFERENCES

[1] Aronow, P., and Samii, C. Estimating average causal effects under interference between units. Manuscript, 2013.

[5] The differences between the variance of a difference in means from a pre-stratified design and a post-stratified estimator with a unstratified design is of order $1/n^2$ [26]. This difference is thus of little importance for large experiments.

[2] Bakshy, E., and Eckles, D. Uncertainty in online experiments with dependent data: An evaluation of bootstrap methods. In *Proc. of the 19th ACM SIGKDD conference on knowledge discovery and data mining*, ACM (2013).

[3] Bakshy, E., Eckles, D., Yan, R., and Rosenn, I. Social influence in social advertising: Evidence from field experiments. In *Proc. of the 13th ACM Conference on Electronic Commerce*, ACM (2012), 146–161.

[4] Bakshy, E., Rosenn, I., Marlow, C., and Adamic, L. The role of social networks in information diffusion. In *Proc. of the 21st international conference on World Wide Web*, ACM (2012), 519–528.

[5] Bareinboim, E., and Pearl, J. Transportability of causal effects: Completeness results. In *Proc. of the Twenty-Sixth National Conference on Artificial Intelligence*, AAAI (2012).

[6] Beenen, G., Ling, K., Wang, X., Chang, K., Frankowski, D., Resnick, P., and Kraut, R. E. Using social psychology to motivate contributions to online communities. In *Proc. of the 2004 ACM conference on Computer supported cooperative work*, CSCW '04, ACM (2004), 212–221.

[7] Bond, R. M., Fariss, C. J., Jones, J. J., Kramer, A. D. I., Marlow, C., Settle, J. E., and Fowler, J. H. A 61-million-person experiment in social influence and political mobilization. *Nature 489*, 7415 (2012), 295–298.

[8] Box, G. E., Hunter, J. S., and Hunter, W. G. *Statistics for Experimenters: Design, Innovation, and Discovery*, vol. 13. Wiley Online Library, 2005.

[9] Bryan, C. J., Walton, G. M., Rogers, T., and Dweck, C. S. Motivating voter turnout by invoking the self. *Proc. of the National Academy of Sciences 108*, 31 (2011), 12653–12656.

[10] Burke, M., Marlow, C., and Lento, T. Feed me: Motivating newcomer contribution in social network sites. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, ACM (2009), 945–954.

[11] Cameron, A., Gelbach, J., and Miller, D. Robust inference with multi-way clustering. *Journal of Business & Economic Statistics 29*, 2 (2011), 238–249.

[12] Crook, T., Frasca, B., Kohavi, R., and Longbotham, R. Seven pitfalls to avoid when running controlled experiments on the web. In *Proc. of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM (2009), 1105–1114.

[13] Deaton, A. Instruments, randomization, and learning about development. *Journal of Economic Literature* (2010), 424–455.

[14] Farahat, A., and Bailey, M. C. How effective is targeted advertising? In *Proc. of the 21st international conference on World Wide Web*, ACM (2012), 111–120.

[15] Gelman, A. Analysis of variance — why it is more important than ever. *The Annals of Statistics 33*, 1 (2005), 1–53.

[16] Gerber, A. S., and Green, D. P. *Field Experiments: Design, Analysis, and Interpretation*. WW Norton, 2012.

[17] Holland, P. W. Causal inference, path analysis, and recursive structural equations models. *Sociological Methodology 18* (1988), 449–484.

[18] Kohavi, R., Deng, A., Frasca, B., Longbotham, R., Walker, T., and Xu, Y. Trustworthy online controlled experiments: Five puzzling outcomes explained. In *Proc. of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM (2012), 786–794.

[19] Kohavi, R., Longbotham, R., Sommerfield, D., and Henne, R. Controlled experiments on the web: Survey and practical guide. *Data Mining and Knowledge Discovery 18*, 1 (2009), 140–181.

[20] Kulkarni, C., and Chi, E. All the news that's fit to read: a study of social annotations for news reading. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2013), 2407–2416.

[21] Lewis, R. A., Rao, J. M., and Reiley, D. H. Here, there, and everywhere: Correlated online behaviors can lead to overestimates of the effects of advertising. In *Proc. of the 20th international conference on World wide web*, ACM (2011), 157–166.

[22] Li, L., Chu, W., Langford, J., and Schapire, R. E. A contextual-bandit approach to personalized news article recommendation. In *Proc. of the 19th international conference on World wide web*, ACM (2010), 661–670.

[23] Manzi, J. *Uncontrolled: The Surprising Payoff of Trial-and-Error for Business, Politics, and Society*. Basic Books, 2012.

[24] Mao, A., Chen, Y., Gajos, K. Z., Parkes, D., Procaccia, A. D., and Zhang, H. Turkserver: Enabling synchronous and longitudinal online experiments. *Proc. HCOMP '12* (2012).

[25] Mason, W., and Suri, S. Conducting behavioral research on Amazon's Mechanical Turk. *Behavior research methods 44*, 1 (2012), 1–23.

[26] Miratrix, L. W., Sekhon, J. S., and Yu, B. Adjusting treatment effect estimates by post-stratification in randomized experiments. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 75*, 2 (2013), 369–396.

[27] Morgan, S. L., and Winship, C. *Counterfactuals and Causal Inference: Methods and Principles for Social Research.* Cambridge University Press, July 2007.

[28] Neilson, J. Putting A/B testing in its place, 2005. http://www.nngroup.com/articles/putting-ab-testing-in-its-place.

[29] Rubin, D. B. Statistics and causal inference: Comment: Which ifs have causal answers. *Journal of the American Statistical Association 81*, 396 (1986), 961–962.

[30] Scott, S. L. A modern Bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry 26*, 6 (2010), 639–658.

[31] Shadish, W. R., and Cook, T. D. The renaissance of field experimentation in evaluating interventions. *Annual Review of Psychology 60*, 1 (Jan. 2009), 607–629.

[32] Tang, D., Agarwal, A., O'Brien, D., and Meyer, M. Overlapping experiment infrastructure: More, better, faster experimentation. In *Proc. of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM (2010), 17–26.

[33] Taylor, S. J., Bakshy, E., and Aral, S. Selection effects in online sharing: Consequences for peer adoption. In *Proc. of the Fourteenth ACM Conference on Electronic Commerce*, EC '13, ACM (2013), 821–836.

[34] Ugander, J., Karrer, B., Backstrom, L., and Kleinberg, J. M. Graph cluster randomization: Network exposure to multiple universes. In *Proc. of KDD*, ACM (2013).

[35] Watts, D. *Everything Is Obvious: *Once You Know the Answer*. Crown Publishing Group, 2011.