Designing and Deploying Online Field Experiments

Eytan Bakshy*
Dean Eckles*
Michael S. Bernstein**

*Facebook
**Stanford

WWW 2014
April 10, 2014
Outline

• Motivation
  • Why run experiments?
  • Problems with running experiments
• Designing experiments with PlanOut
• Deploying experiments:
  • Logging and analysis
  • Management and iterative experimentation
• Discussion
Motivation
Goals
Test complete alternatives
Explore a design space
Understand tradeoffs

Understand tradeoffs

Passive sharing (old sharing model)

Active sharing (new sharing model)

Attribute outcomes to causes

Non-social ad

Social ad

Attribute outcomes to causes

Experiments can be hard
Organizational problems

• Multiple simultaneous experiments
  • Sometimes need to iterate
• Not clear how to manage changes to experiments
  • Changing experiments breaks randomization
Engineering problems

- Experimental logic is embedded in application code
- Distributed logic makes experiments brittle
- Difficult to understand for those who didn’t write the code
- Difficult to run experiments on multiple platforms
Parameterization of user experiences

- Feed width
- Font size
- Image width
- Padding
- Number of social cues
PlanOut
PlanOut scripts are high-level descriptions of randomized parameterizations
The PlanOut idea

• User experiences are parameterized by experimental assignments

• PlanOut scripts describe assignment procedures

• Experiments are PlanOut scripts plus a population

• Parallel or follow-on experiments are centrally managed
Sample PlanOut script

```python
button_color = uniformChoice(
    choices=['#ff0000', '#00ff00'],
    unit=userid);

button_text = uniformChoice(
    choices=['I'm voting', 'I'm a voter'],
    unit=userid);
```

2x2 factorial design
Compiled PlanOut code

```json
{
  "op": "seq",
  "seq": [
    {
      "op": "set",
      "var": "button_color",
      "value": {
        "choices": {
          "op": "array",
          "values": [
            "#ff0000",
            "#00ff00"
          ]
        },
        "unit": {
          "op": "get",
          "var": "userid"
        }
      }
    },
    {
      "op": "uniformChoice"
    }
  ],
  "op": "set",
  "var": "button_text",
  "value": {
    "choices": {
      "op": "array",
      "values": [
        "I'm voting",
        "I'm a voter"
      ]
    },
    "unit": {
      "op": "get",
      "var": "userid"
    }
  }
}
```
PlanOut code is portable
PlanOut code can be generated by GUIs

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
How does it work?
e = getExp('share_dialog', userid=20)
button_color = e.get('button_color')

retrieve experiment script

button_color = weightedChoice(
choices=['#ff0000', '#00ff00'],
weights=[0.2, 0.8], unit=userid);

construct hash string

share_dialog.button_color.20

f(SHA1(str))

"#ff0000"
Examples of PlanOut experiments
What is the marginal effect of social cues on an action?
Social cues experiment

num_cues = randomInteger(
    min=1, max=min(length(liking_friends), 3),
    unit=[userid, pageid]);
num_cues = randomInteger(
    min=1, max=min(length(liking_friends), 3),
    unit=[userid, adid]);

friends_shown = sample(
    choices=liking_friends, draws=num_cues,
    unit=[userid, adid]);
How can we increase voting?

Can we motivate voter turnout by invoking the self?
Tell friends you're voting in the 2012 Election and find out where to vote.

I'm a Voter

Find My Polling Place

Adam, Adam and 4 others are voters in this election.

Tell friends you're voting in the 2012 Election and find out where to vote.

I'm Voting

Find My Polling Place
Banner exposure

User action

Intensity of social cues

News Feed exposure
<table>
<thead>
<tr>
<th>Has banner</th>
<th>No banner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Has feed stories</td>
<td>2%</td>
</tr>
<tr>
<td>No feed stories</td>
<td>95%</td>
</tr>
<tr>
<td>(sum)</td>
<td>97%</td>
</tr>
</tbody>
</table>

```python
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);
```
To what extent does feedback affect content production?
Fact:
Collapsing comment boxes modulates feedback

Expanded comment box

Collapsed comment box

Higher interaction rate

Lower interaction rate
Effect of feedback on content production

prob_collapse = randomFloat(
    min=0.0, max=1.0, unit=sourceid);
Effect of feedback on content production

```
probCollapse = randomFloat(
    min=0.0, max=1.0, unit=sourceid);

collapse = bernoulliTrial(
    p=probCollapse, unit=[storyid, viewerid]);
```

- Simple: probCollapse vs content production
- Fancy (2SLS): 
  production ~ comments; comments ~ probCollapse
Logging and analysis
e = getExp('share_dialog', userid=20)

button_color = e.get('button_color')

trigger auto exposure log

```json
{
  "inputs": {
    "userid": 20
  },
  "name": "sharing_dialog",
  "params": {
    "button_color": "#ff0000",
    "button_text": "Share with others"
  },
  "time": 1396507677,
  "salt": "sharing_dialog",
  "event": "exposure"
}
```
```python
e = getExp('share_dialog', userid=20)

e.log_event('button_click')

{  
  "inputs": {  
    "userid": 20
  },  
  "name": "sharing_dialog",  
  "params": {  
    "button_color": "#ff0000",  
    "button_text": "Share with others"
  },  
  "time": 1396507677,  
  "salt": "sharing_dialog",  
  "event": "button_click"
}
```
outcomes joined with exposure log

exposure log
metrics
event log

Hive + R

automatic analysis

SQL + D3

graphical interfaces

e.g. relative change + confidence intervals

results can be looked up by experiment name
Managing and analyzing related experiments
Namespaces: a way of managing experiments
e = getExp('share_dialog', userid=20)

button_color = e.get('button_color')

Hash userid to a NS-specific segment number

segment to experiment mapping

Exp 1  Exp 2  Default value store

log + return values  return values
Deploying iterative experiments

- Week 1: launch initial PlanOut script with many conditions
- Week 2: launch same script with more segments
- Week 4: launch new script with fewer conditions, more segments

segments
Best practices for iterative experiments

- Use namespaces for all experiments
- Changes to experiments done through launching new experiments
- Follow-on experiments analyzed separately
- Pool results of related experiments with care*

Recap

- Experiments are useful for knowledge building
- PlanOut supports good experiments
  - Focuses experimenters on parameters
  - Makes it easy to run complex studies
- Management frameworks prevent you from shooting yourself in the foot
Use PlanOut

http://facebook.github.io/planout

> easy_install planout
Thank you!

• Coauthors
  Dean Eckles
  Michael Bernstein

• Special thanks: Daniel Ting, Wojtek Galuba, Breno Roberto, Wesley May