Outbreak Detection in Networks

CS224W: Analysis of Networks
Jure Leskovec, Stanford University
http://cs224w.stanford.edu
(1) New problem: Outbreak detection
(2) Develop an approximation algorithm
   - It is a submodular opt. problem!
(3) Speed-up greedy hill-climbing
   - Valid for optimizing general submodular functions
     (i.e., also works for influence maximization)
(4) Prove a new “data dependent” bound on the solution quality
   - Valid for optimizing any submodular function
     (i.e., also works for influence maximization)
Detecting Contamination Outbreaks

- Given a real city water distribution network
- And data on how contaminants spread in the network
- Detect the contaminant as quickly as possible
- Problem posed by the US Environmental Protection Agency
Detecting Information Outbreaks

Which users/news sites should one follow to detect cascades as effectively as possible?
Detecting Information Outbreaks

Detect **blue** & **yellow stories** soon but miss the **red story**.

**Want to read things before others do.**

Detect **all stories** but **late**.
Both of these two are instances of the same underlying problem!

Given a dynamic process spreading over a network we want to select a set of nodes to detect the process effectively.

Many other applications:
- Epidemics
- Influence propagation
- Network security
Water Network: Utility

- **Utility of placing sensors:**
  - Water flow dynamics, demands of households, ...
- **For each subset** $S \subseteq V$ **compute utility** $f(S)$

![Diagram showing high impact outbreak, medium impact outbreak, and low impact outbreak with sensors S1, S2, S3, S4 and their impact on contamination.]

- High sensing “quality” (e.g., $f(S) = 0.9$)
- Low sensing “quality” (e.g., $f(S) = 0.01$)

Sensor reduces impact through early detection!
Given:
- Graph \( G(V, E) \)
- Data about how outbreaks spread over the \( G \):
  - For each outbreak \( i \) we know the time \( T(u, i) \) when outbreak \( i \) contaminates node \( u \)

Water distribution network (physical pipes and junctions)

Simulator of water consumption & flow (built by Mech. Eng. people)
We simulate the contamination spread for every possible location.
Problem Setting: News

Given:
- Graph $G(V, E)$
- Data about how outbreaks spread over the $G$:
  - For each outbreak $i$ we know the time $T(u, i)$ when outbreak $i$ contaminates node $u$

The network of newsmedia

Traces of the information flow and identify influence sets
Collect lots of articles and trace them to obtain data about information flow from a given news site.
Problem Setting

**Given:**
- Graph $G(V, E)$
- Data on **how outbreaks spread over the $G$:**
  - For each outbreak $i$ we know the time $T(u, i)$ when outbreak $i$ contaminates node $u$

**Goal:** Select a subset of nodes $S$ that maximizes the expected **reward**:

$$ \max_{S \subseteq V} f(S) = \sum_{i} P(i) f_i(S) $$

subject to: $\text{cost}(S) < B$

$P(i)$... probability of outbreak $i$ occurring.
$f(i)$... reward for detecting outbreak $i$ using sensors $S$. 
Two Parts to the Problem

- **Reward (one of the following three):**
  - (1) Minimize time to detection
  - (2) Maximize number of detected propagations
  - (3) Minimize number of infected people

- **Cost (context dependent):**
  - Reading big blogs is more time consuming
  - Placing a sensor in a remote location is expensive

Monitoring **blue** node saves more people than monitoring the **green** node

\( f(S) \)
Objective functions:

1) **Time to detection** (DT)
   - How long does it take to detect a contamination?
   - **Penalty for detecting at time** $t$: $\pi_i(t) = t$

2) **Detection likelihood** (DL)
   - How many contaminations do we detect?
   - **Penalty for detecting at time** $t$: $\pi_i(t) = 0, \pi_i(\infty) = 1$
     - Note, this is binary outcome: we either detect or not

3) **Population affected** (PA)
   - How many people drank contaminated water?
   - **Penalty for detecting at time** $t$: $\pi_i(t) = \{\# \text{ of infected nodes in outbreak } i \text{ by time } t\}$.

**Observation:**

In all cases detecting sooner does not hurt!
We define $f_i(S)$ as penalty reduction:

$$f_i(S) = \pi_i(\emptyset) - \pi_i(T(S, i))$$

- **Observation:** **Diminishing returns**

Placement $S=\{x_1, x_2\}$

Adding $s'$ helps a lot

New sensor:

Placement $S'=\{x_1, x_2, x_3, x_4\}$

Adding $s'$ helps very little
Objective functions are Submodular

- **Claim:** For all $A \subseteq B \subseteq V$ and sensors $s \in V \setminus B$
  \[ f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B) \]

- **Proof:** All our objectives are submodular
  - Fix cascade/outbreak $i$
  - Show $f_i(A) = \pi_i(\infty) - \pi_i(T(A, i))$ is submodular
  - Consider $A \subseteq B \subseteq V$ and sensor $x \in V \setminus B$
  - When does node $s$ detect cascade $i$?
    - We analyze 3 cases based on when $x$ detects outbreak $i$
    - (1) $T(B, i) < T(A, i) < T(x, i)$: $x$ detects late, nobody benefits:
      \[ f_i(A \cup \{x\}) = f_i(A), \text{ also } f_i(B \cup \{x\}) = f_i(B) \text{ and so} \]
      \[ f_i(A \cup \{x\}) - f_i(A) = 0 = f_i(B \cup \{x\}) - f_i(B) \]
Objective functions are Submodular

- **Proof (contd.):**
  - (2) $T(B, i) \leq T(x, i) \leq T(A, i)$: $x$ detects after $B$ but before $A$ 
x detects sooner than any node in $A$ but after all in $B$.
So $x$ only helps improve the solution $A$ (but not $B$)

\[
f_i(A \cup \{x\}) - f_i(A) \geq 0 = f_i(B \cup \{x\}) - f_i(B)
\]

- (3) $T(x, i) < T(B, i) < T(A, i)$: $x$ detects early

\[
f_i(A \cup \{x\}) - f_i(A) = [\pi_i(\infty) - \pi_i(T(x, i))] - f_i(A) \geq
[\pi_i(\infty) - \pi_i(T(x, i))] - f_i(B) = f_i(B \cup \{x\}) - f_i(B)
\]

- Inequality is due to non-decreasingness of $f_i(\cdot)$, i.e., $f_i(A) \leq f_i(B)$

- **So, $f_i(\cdot)$ is submodular!**
- **So, $f(\cdot)$ is also submodular**

\[
f(S) = \sum_i P(i) f_i(S)
\]
What do we know about optimizing submodular functions?

- Hill-climbing (i.e., greedy) is near optimal: \((1 - \frac{1}{e}) \cdot OPT\)

But:

- (1) This only works for unit cost case! (each sensor costs the same)
  - For us each sensor \(s\) has cost \(c(s)\)
- (2) Hill-climbing algorithm is slow
  - At each iteration we need to re-evaluate marginal gains of all nodes
  - Runtime \(O(|V| \cdot K)\) for placing \(K\) sensors
CELF: Algorithm for optimizing submodular functions under cost constraints
Consider the following algorithm to solve the outbreak detection problem:

**Hill-climbing that ignores cost**
- Ignore sensor cost $c(s)$
- Repeatedly select sensor with highest marginal gain
- Do this until the budget is exhausted

**Q: How well does this work?**

**A: It can fail arbitrarily badly! 😞**
- There exists a problem setting where the hill-climbing solution is arbitrarily far from OPT
- Next we come up with an example
Problem 1: Ignoring Cost

- **Bad example when we ignore cost:**
  - $n$ sensors, budget $B$
  - $s_1$: reward $r$, cost $B$
  - $s_2 \ldots s_n$: reward $r - \varepsilon$, $c = 1$
  - Hill-climbing always prefers more expensive sensor $s_1$ with reward $r$ (and exhausts the budget).
    It never selects cheaper sensors with reward $r - \varepsilon$
  \[\rightarrow\text{For variable cost it can fail arbitrarily badly!}\]
- **Idea:** What if we optimize benefit-cost ratio?

\[
s_i = \arg \max_{s \in V} \frac{f(A_{i-1} \cup \{s\}) - f(A_{i-1})}{c(s)}
\]

Greedily pick sensor $s_i$ that maximizes benefit to cost ratio.
Problem 2: Benefit-Cost

- Benefit-cost ratio can also fail arbitrarily badly!
- **Consider**: budget $B$:
  - 2 sensors $s_1$ and $s_2$:
    - Costs: $c(s_1) = \varepsilon$, $c(s_2) = B$
    - Benefit (only 1 cascade): $f(s_1) = 2\varepsilon$, $f(s_2) = B$
  - **Then benefit-cost ratio is**:
    - $f(s_1)/c(s_1) = 2$ and $f(s_2)/c(s_2) = 1$
  - So, we first select $s_1$ and then can not afford $s_2$
  - $\Rightarrow$ We get reward $2\varepsilon$ instead of $B$! Now send $\varepsilon \to 0$
    and we get an **arbitrarily bad solution**!

This algorithm incentivizes choosing nodes with very low cost, even when slightly more expensive ones can lead to much better global results.
Solution: CELF Algorithm

- **CELF** (Cost-Effective Lazy Forward-selection)
  A two pass greedy algorithm:
  - Set (solution) $S'$: Use benefit-cost greedy
  - Set (solution) $S''$: Use unit-cost greedy
  - Final solution: $S = \arg \max (f(S'), f(S''))$

- **How far is CELF from (unknown) optimal solution?**

- **Theorem:** CELF is near optimal [Krause&Guestrin, ‘05]
  - CELF achieves $\frac{1}{2}(1-1/e)$ factor approximation!

*This is surprising:* We have two clearly suboptimal solutions, but taking best of the two is guaranteed to give a near-optimal solution.
Speeding-up Hill-Climbing: Lazy Evaluations
What do we know about optimizing submodular functions?

- Hill-climbing (i.e., greedy) is near optimal (that is, $(1 - \frac{1}{e}) \cdot OPT$)

But:

- (2) Hill-climbing algorithm is slow!
  - At each iteration we need to re-evaluate marginal gains of all nodes
  - Runtime $O(|V| \cdot K)$ for placing $K$ sensors
Speeding up Hill-Climbing

- **In round $i + 1$:** So far we picked $S_i = \{s_1, \ldots, s_i\}$
  - Now pick $s_{i+1} = \arg \max_u f(S_i \cup \{u\}) - f(S_i)$
    - This our old friend – greedy hill-climbing algorithm.
      It maximizes the “marginal gain”
      $$\delta_i(u) = f(S_i \cup \{u\}) - f(S_i)$$

- **By submodularity property:**
  $$f(S_i \cup \{u\}) - f(S_i) \geq f(S_j \cup \{u\}) - f(S_j) \text{ for } i < j$$

- **Observation: By submodularity:**
  For every $u$
  $$\delta_i(u) \geq \delta_j(u) \text{ for } i < j \text{ since } S_i \subset S_j$$
  
  **Marginal benefits $\delta_i(u)$ only shrink!**
  (as i grows)

Activating node $u$ in step $i$ helps more than activating it at step $j$ ($j > i$)
**Idea:**

- Use $\delta_i$ as upper-bound on $\delta_j$ ($j > i$)

**Lazy hill-climbing:**

- Keep an ordered list of marginal benefits $\delta_i$ from previous iteration
- Re-evaluate $\delta_i$ only for top node
- Re-order and prune

$$f(S \cup \{u\}) - f(S) \geq f(T \cup \{u\}) - f(T) \quad S \subseteq T$$

Marginal gain

- $S_1 = \{a\}$
- $a$
- $b$
- $c$
- $d$
- $e$
Lazy Hill Climbing

- **Idea:**
  - Use $\delta_i$ as upper-bound on $\delta_j$ ($j > i$)

- **Lazy hill-climbing:**
  - Keep an ordered list of marginal benefits $\delta_i$ from previous iteration
  - Re-evaluate $\delta_i$ **only** for top node
  - Re-order and prune

$$f(S \cup \{u\}) - f(S) \geq f(T \cup \{u\}) - f(T) \quad S \subseteq T$$
Lazy Hill Climbing

- **Idea:**
  - Use $\delta_i$ as upper-bound on $\delta_j$ ($j > i$)
- **Lazy hill-climbing:**
  - Keep an ordered list of marginal benefits $\delta_i$ from previous iteration
  - Re-evaluate $\delta_i$ only for top node
  - Re-order and prune

\[
f(S \cup \{u\}) - f(S) \geq f(T \cup \{u\}) - f(T) \quad S \subseteq T\]
CELF: Scalability

- CELF (using Lazy evaluation) runs 700 times faster than greedy hill-climbing algorithm

- CELF... raw CELF
- CELF+bounds ... CELF together with computing the data-dependent solution quality bound
Data Dependent Bound on the Solution Quality
Back to the solution quality!

The \((1-1/e)\) bound for submodular functions is the worst case bound (worst over all possible inputs)

Data dependent bound:
- Value of the bound depends on the input data
  - On “easy” data, hill climbing may do better than 63%
- Can we say something about the solution quality when we know the input data?
Data Dependent Bound

- Suppose $S$ is some solution to $f(S)$ s.t. $|S| \leq k$
  - $f(S)$ is monotone & submodular
- Let $OPT = \{t_1, ..., t_k\}$ be the OPT solution
- For each $u$ let $\delta(u) = f(S \cup \{u\}) - f(S)$
- Order $\delta(u)$ so that $\delta(1) \geq \delta(2) \geq \cdots$
- Then: $f(OPT) \leq f(S) + \sum_{i=1}^{k} \delta(i)$

- Note:
  - This is a data dependent bound ($\delta(i)$ depends on input data)
  - Bound holds for any algorithm
    - Makes no assumption about how $S$ was computed
  - For some inputs it can be very “loose” (worse than 63%)
Claim:

- For each $u$ let $\delta(u) = f(S \cup \{u\}) - f(S)$
- Order $\delta(u)$ so that $\delta(1) \geq \delta(2) \geq \cdots$

Then: $f(OPT) \leq f(S) + \sum_{i=1}^{k} \delta(i)$

Proof:

- $f(OPT) \leq f(OPT \cup S)$
- $= f(S) + f(OPT \cup S) - f(S)$
- $\leq f(S) + \sum_{i=1}^{k} [f(S \cup \{t_i\}) - f(S)]$
- $= f(S) + \sum_{i=1}^{k} \delta(t_i)$

Instead of taking $t_i \in OPT$ (of benefit $\delta(t_i)$), we take the best possible element ($\delta(i)$)

$\leq f(S) + \sum_{i=1}^{k} \delta(i) \Rightarrow f(OPT) \leq f(S) + \sum_{i=1}^{k} \delta(i)$
Case Study: Water distribution network & blogs
Case Study: Water Network

- Real metropolitan area water network
  - $V = 21,000$ nodes
  - $E = 25,000$ pipes

- Use a cluster of 50 machines for a month
- Simulate 3.6 million epidemic scenarios (random locations, random days, random time of the day)
Bounds on the Optimal Solution

Data-dependent bound is much tighter (gives more accurate estimate of alg. performance)
Water: Heuristic Placement

- Placement heuristics perform much worse

<table>
<thead>
<tr>
<th>Author</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CELF</td>
<td>26</td>
</tr>
<tr>
<td>Sandia</td>
<td>21</td>
</tr>
<tr>
<td>U Exter</td>
<td>20</td>
</tr>
<tr>
<td>Bentley systems</td>
<td>19</td>
</tr>
<tr>
<td>Technion (1)</td>
<td>14</td>
</tr>
<tr>
<td>Bordeaux</td>
<td>12</td>
</tr>
<tr>
<td>U Cyprus</td>
<td>11</td>
</tr>
<tr>
<td>U Guelph</td>
<td>7</td>
</tr>
<tr>
<td>U Michigan</td>
<td>4</td>
</tr>
<tr>
<td>Michigan Tech U</td>
<td>3</td>
</tr>
<tr>
<td>Malcolm</td>
<td>2</td>
</tr>
<tr>
<td>Proteo</td>
<td>2</td>
</tr>
<tr>
<td>Technion (2)</td>
<td>1</td>
</tr>
</tbody>
</table>

Battle of Water Sensor Networks competition
Different objective functions give different sensor placements

Population affected

Detection likelihood
Here CELF is much faster than greedy hill-climbing!

- (But there might be datasets/inputs where the CELF will have the same running time as greedy hill-climbing)
Question...

= I have 10 minutes. Which news sites should I read to be most up to date?

= Who are the most influential news sites?
Detecting Information Outbreaks

Detect blue & yellow soon but miss red.

Want to read things before others do.

Detect all stories but late.
Case study 2: Cascades in blogs

- Crawled 45,000 blogs for 1 year
- Obtained 10 million news posts
- And identified 350,000 cascades
- Cost of a blog is the number of posts it has
Online bound turns out to be much tighter!

Based on the plot below: 87% instead of 32.5%
Heuristics perform much worse!
One really needs to perform the optimization
Blogs: Cost of a Blog

- CELF has 2 sub-algorithms. Which wins?
- **Unit cost:**
  - CELF picks large popular blogs
- **Cost-benefit:**
  - Cost proportional to the number of posts
- We can do much better when considering costs
Blogs: Cost of a Blog

- **Problem:** Then CELF picks *lots of small blogs* that participate in few cascades
- We pick best solution that interpolates between the costs
- We can get good solutions with *few blogs and few posts*

**Each curve represents a set of solutions S with the same final reward f(S)**
We want to generalize well to future (unknown) cascades

Limiting selection to bigger blogs improves generalization!
**Blogs: Scalability**

- **CELF** runs **700** times faster than simple hill-climbing algorithm

---

[Leskovec et al., KDD ‘07]
Outbreak detection problem in networks

Different ways to formalize objective functions
  All are submodular

Lazy-Greedy algorithm for optimizing submodular functions

CELF algorithm that combines 2 versions of Lazy-Greedy

Data-dependent bound on the solution quality