Outbreak Detection in Networks

Thanks to Jure Leskovec, Stanford and Panayiotis Tsaparas, Univ. of Ioannina for slides
Plan for Today

- (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
Which blogs should one read to detect cascades as effectively as possible?
Detecting Information Outbreaks

Detect blue & yellow soon but miss red.

Want to read things before others do.

Detect all stories but late.
Both of these two are an instance 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** $f(S)$

High sensing quality $f(S) = 0.9$

Low sensing quality $f(S) = 0.01$
Problem Setting: Contamination

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

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

The network of the blogosphere

Traces of the information flow
Collect lots of blogs posts and trace hyperlinks to obtain data about information flow from a given blog.
Problem Setting

**Given:**
- Graph $G(V, E)$
- Data on how outbreaks spread over the $G$:
  - For each outbreak $i$ we know the time $T(i, u)$ 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: $cost(S) < B$
Two Parts to the Problem

- **Reward**
  - (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

$\text{monitoring } i$
Objective functions are Submodular

- **Objective functions:**
  - 1) **Time to detection** (DT)
    - How long does it take to detect a contamination?
    - Penalty for detecting at time $t$: $\pi_i(t) = \min\{t, T_{max}\}$
  - 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!

\[ f_i(S) \text{ is penalty reduction: } f_i(S) = \pi_i(\emptyset) - \pi_i(S) \]
Observation: **Diminishing returns**

Placement $S = \{s_1, s_2\}$

New sensor: $s'$

Placement $S' = \{s_1, s_2, s_3, s_4\}$

Adding $s'$ helps a lot

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

Claim: For all $A \subseteq B \subseteq V$ and sensors $s \in V\setminus B$

$$f(A \cup \{s\}) - f(A) \geq f(B \cup \{s\}) - f(B)$$

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 $s \in V\setminus B$
- When does node $s$ detect cascade $i$?

  - We analyze 3 cases based on when $s$ detects outbreak $i$
  - (1) $T(s, i) \geq T(A, i)$: $s$ detects late, nobody benefits: $f_i(A \cup \{s\}) = f_i(A)$, also $f_i(B \cup \{s\}) = f_i(B)$ and so $f_i(A \cup \{s\}) - f_i(A) = 0 = f_i(B \cup \{s\}) - f_i(B)$
Objective functions are Submodular

Proof (contd.):

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

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

- (3) \( T(s, i) < T(B, i) \): \( s \) detects early

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

- 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?

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

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
- 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! 😞**

- Next we come up with an example where Hill-climbing solution is arbitrarily away from OPT
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 \( r - \varepsilon \), cost 1
  - Hill-climbing always prefers more expensive sensor \( s_1 \) with reward \( r \) (and exhausts the budget).
  - It never selects cheaper sensors with reward \( r - \varepsilon \)
  - \( \rightarrow \) 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: $c(s_1) = \varepsilon$, $c(s_2) = B$
    - Only 1 cascade: $f(s_1) = 2\varepsilon$, $f(s_2) = B$
  - Then benefit-cost ratio is:
    - $B/c(s_1) = 2$ and $B/c(s_2) = 1$
  - So, we first select $s_1$ and then can not afford $s_2$
  - We get reward $2\varepsilon$ instead of $B$! Now send $\varepsilon \to 0$ and we get 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: \(S = \text{arg max}(f(S'), f(S''))\)

- 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 the best of them always gives us a near-optimal solution.
Speeding-up Hill-Climbing: Lazy Evaluations
What do we know about optimizing submodular functions?

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

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 is our old friend – greedy hill-climbing algorithm. It maximizes the “marginal benefit”
    \[
    \delta_i(u) = f(S_i \cup \{u\}) - f(S_i)
    \]

- **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 \( u \)
  \[
  \delta_i(u) \geq \delta_j(u) \text{ for } i < j \text{ since } S_i \subseteq S_j
  \]
  **Marginal benefits \( \delta_i(u) \) only shrink!**
  (as \( i \) grows)

Activating node \( u \) in step \( i \) helps more than activating it at step \( j \) (\( j > i \))
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-sort 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-sort and prune

\[
\frac{\text{Marginal gain}}{a \quad S_1 = \{a\}}
\]

\[
\begin{align*}
\text{Marginal gain} & \quad a \\
& \quad b \\
& \quad c \\
& \quad d \\
& \quad e \\
\end{align*}
\]

\[
f(S \cup \{u\}) - f(S) \geq f(T \cup \{u\}) - f(T) \quad S \subseteq T
\]
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-sort and prune

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

Marginal gain

- $a$  
- $d$
- $b$
- $e$
- $c$

$S_1 = \{a\}$

$S_2 = \{a, b\}$
CELF: Scalability

- CELF (using Lazy evaluation) runs 700 times faster than greedy hill-climbing algorithm.
Data Dependent Bound on the Solution Quality
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 \( OPT = \{t_1, \ldots, t_k\} \) be the \( OPT \) solution
- For each \( u \) let \( \delta(u) = f(S \cup \{u\}) - f(S) \)
- Order \( \delta(u) \) so that \( \delta(1) \geq \delta(2) \geq \ldots \)
- Then: \( f(OPT) \leq f(S) + \sum_{i=1}^{k} \delta(i) \)

- Note:
  - This is a data dependent bound (\( \delta(u) \) 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%)
Data Dependent Bound

- **Claim:**
  - For each $u$ let $\delta(u) = f(S \cup \{u\}) - f(S)$
  - Order $\delta(u)$ so that $\delta(1) \geq \delta(2) \geq \ldots$
  - Then: $f(OPT) \leq f(S) + \sum_{i=1}^{k} \delta(i)$

- **Proof:**
  
  \[
  f(OPT) \leq f(OPT \cup S) = f(S) + \sum_{i=1}^{k} [f(S \cup (\text{we proved this last time})]
  \]

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

Solution quality $F(A)$
Higher is better

"Offline"
the $(1-1/e)$ bound

Data-dependent bound

Hill Climbing

Number of sensors placed

0 5 10 15 20

0 0.2 0.4 0.6 0.8 1 1.2 1.4

3/20/2017
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

[w/ Ostfeld et al., J. of Water Resource Planning]
Different objective functions give different sensor placements

Population affected

Detection likelihood
- CELF is 10 times faster than greedy hill-climbing!
= I have 10 minutes. Which blogs should I read to be most up to date?

= Who are the most influential bloggers?
Detecting information outbreaks

Want to read things before others do.

Detect blue & yellow soon but miss red.

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

- Crawled 45,000 blogs for 1 year
- Obtained 10 million 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%
Blogs: Heuristic Selection

- 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
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 $f(S)$. The score for each curve is indicated as $f(S) = 0.4$, $f(S) = 0.3$, and $f(S) = 0.2$. The x-axis represents the number of posts, and the y-axis represents the number of blogs.
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]