Algorithmic Problems in Epidemiology

Anil Vullikanti and Madhav Marathe

Dept. of Computer Science and
Virginia Bioinformatics Institute
Virginia Tech

Questions for a Public Health administrator

- Dynamics of epidemic
  - How many infections?
  - How long will the epidemic last?
- Detecting an outbreak
  - Information from hospitals, pharmacies, media, etc.
  - Sensors placed around the city
- Controlling an outbreak
  - Quarantine people/Close down locations
  - Vaccinate people

Outline

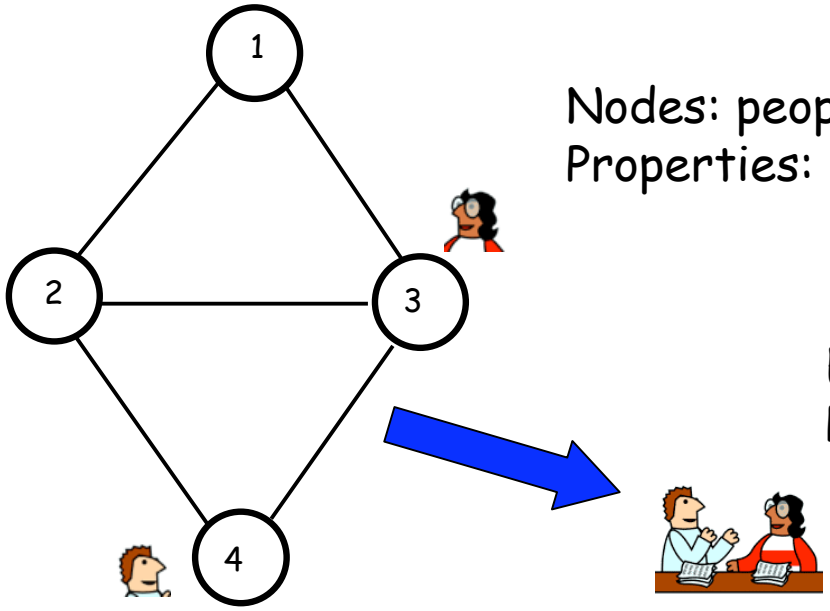
- John Snow and cholera
- A combinatorial model for diseases
- A formulation for the sensor placement problem
- A formulation for the vaccination problem

The first Network Approach to Epidemiology

- Made an implicit network between people and water pumps
- People who drank water from Broad St. pump died
- also proved that cholera spreads through water

"Riddle of the Cholera Outbreak" solved by Dr. John Snow

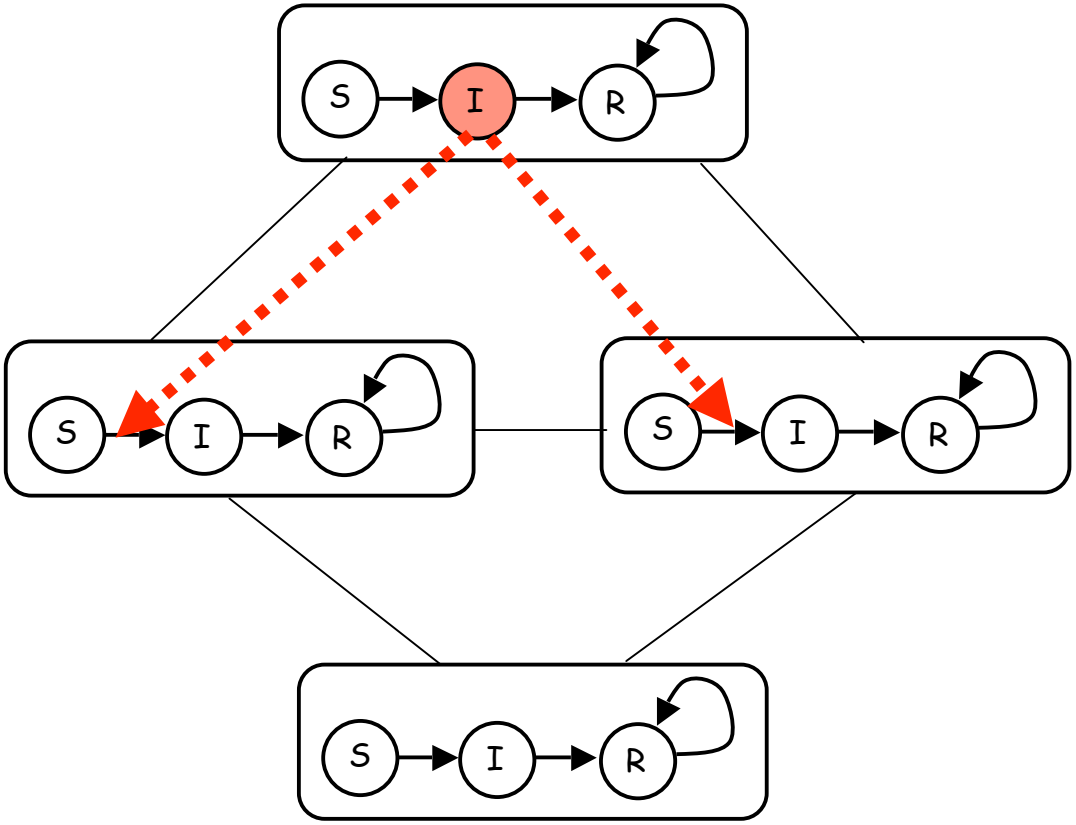
Contact Graph



Nodes: people
Properties: demographics, immunity

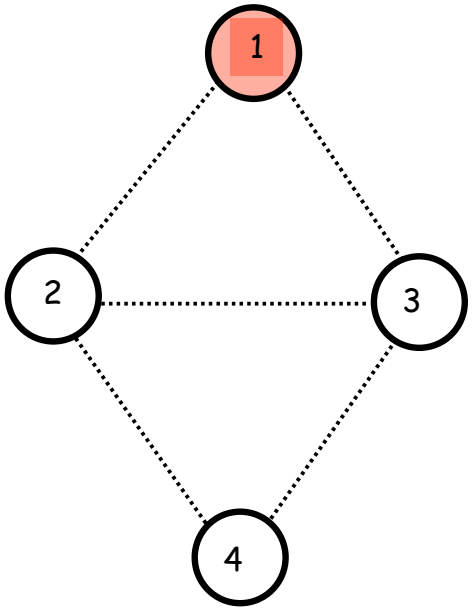
Edges: contacts between people
Properties: duration, nature of contact

Epidemic on network: Communicating FSM Model



Transition probabilities for a node depend on neighbor's states

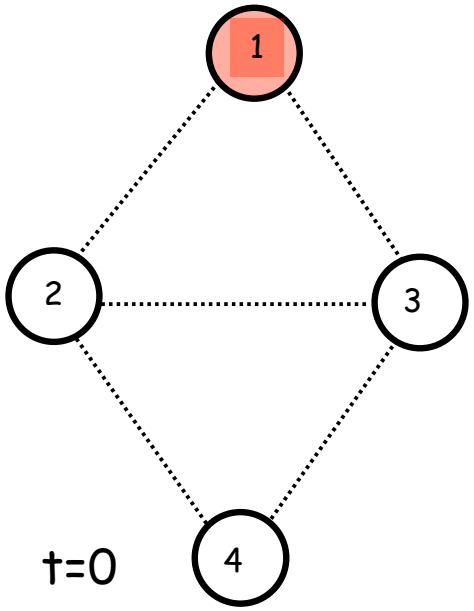
The SIR Disease Model

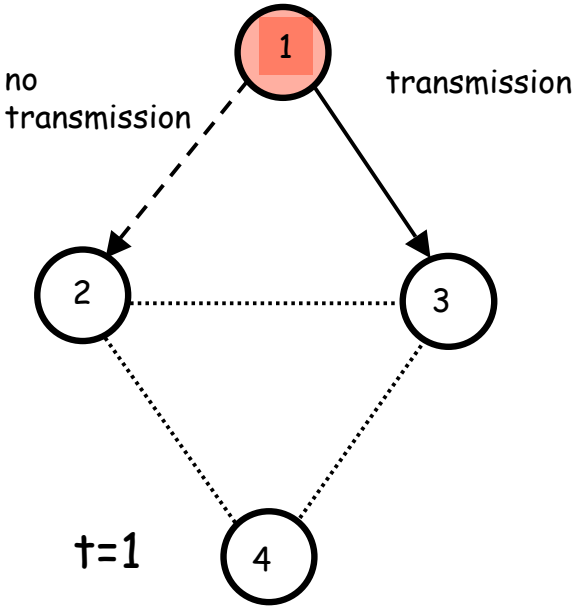


t=0

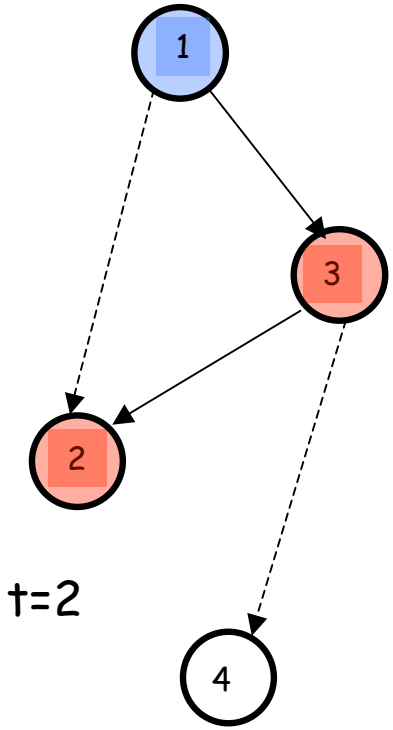
- Discrete SIR process
- Each node remains infected for 2 time steps
- Transmission probability on edge (u,v) = p(u,v)
- Initially node 1 infected

Sample run





Probability =
p(1,3)(1-p(1,2))



Probability =
(1-p(1,2))p(3,2)(1-p(3,4))

Susceptible

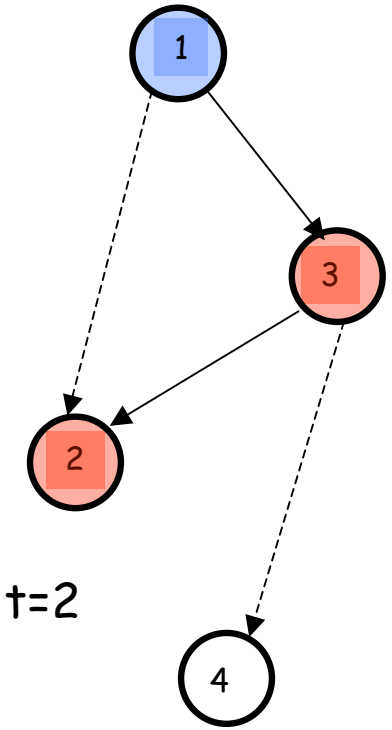
Infected

Recovered

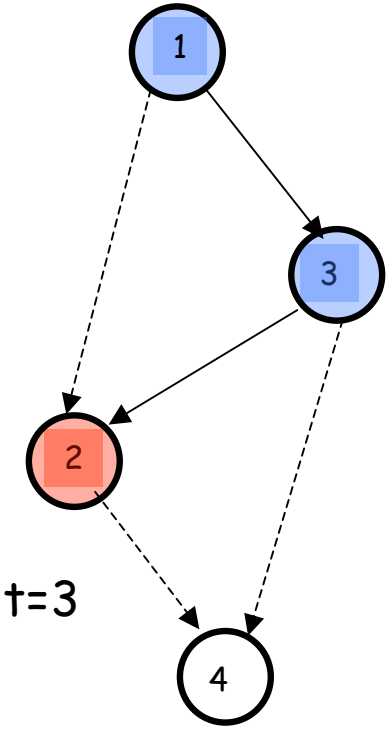
transmission

No transmission

Sample run

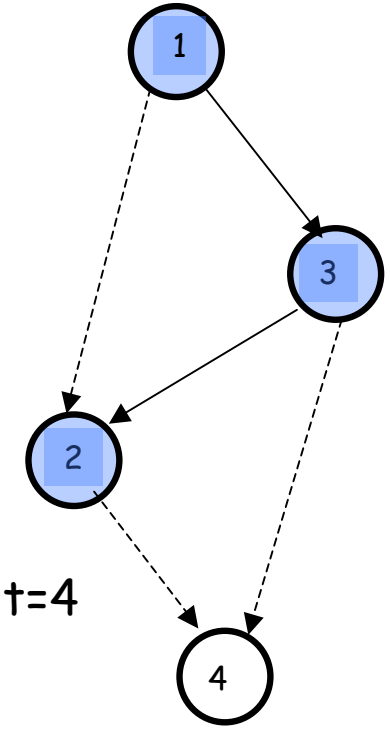


t=2



t=3

Probability =
(1-p(2,4))(1-p(3,4))



t=4

Probability =
(1-p(2,4))

Susceptible

Infected

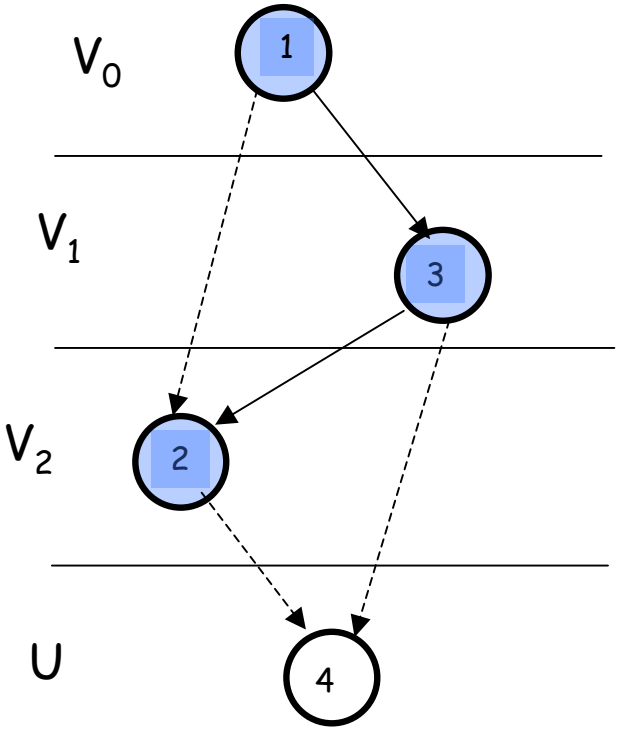
Recovered

transmission

No transmission

All nodes are recovered/susceptible: simulation stops

Formalizing the output of a simulation run



$I = \{ (1,3), (3,2)\}$

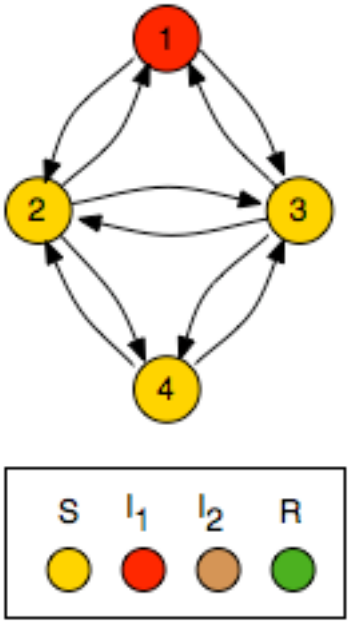
- V_t: set of nodes that got infected at step t
- U: set of uninfected nodes
- I: set of edges on which infection spread (solid edge)
- T: duration of epidemic
- The information in one run of the SIR process on a network is captured by the tuple:

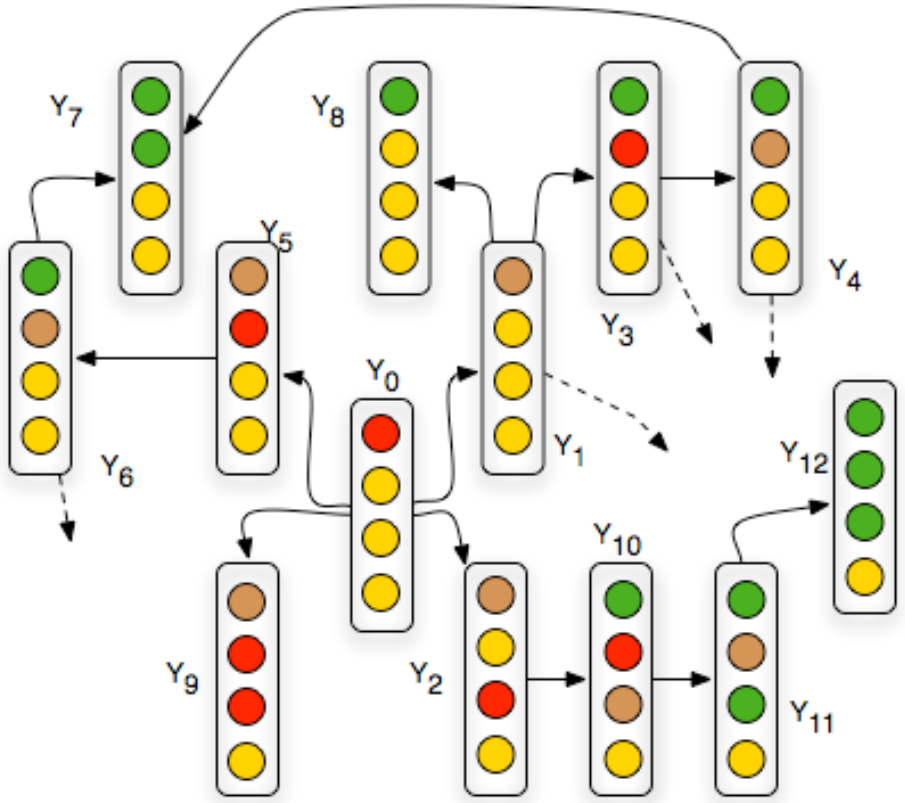
$R = ((V_0, V_1, \ldots, V_T, U), I)$

Probability of this run,

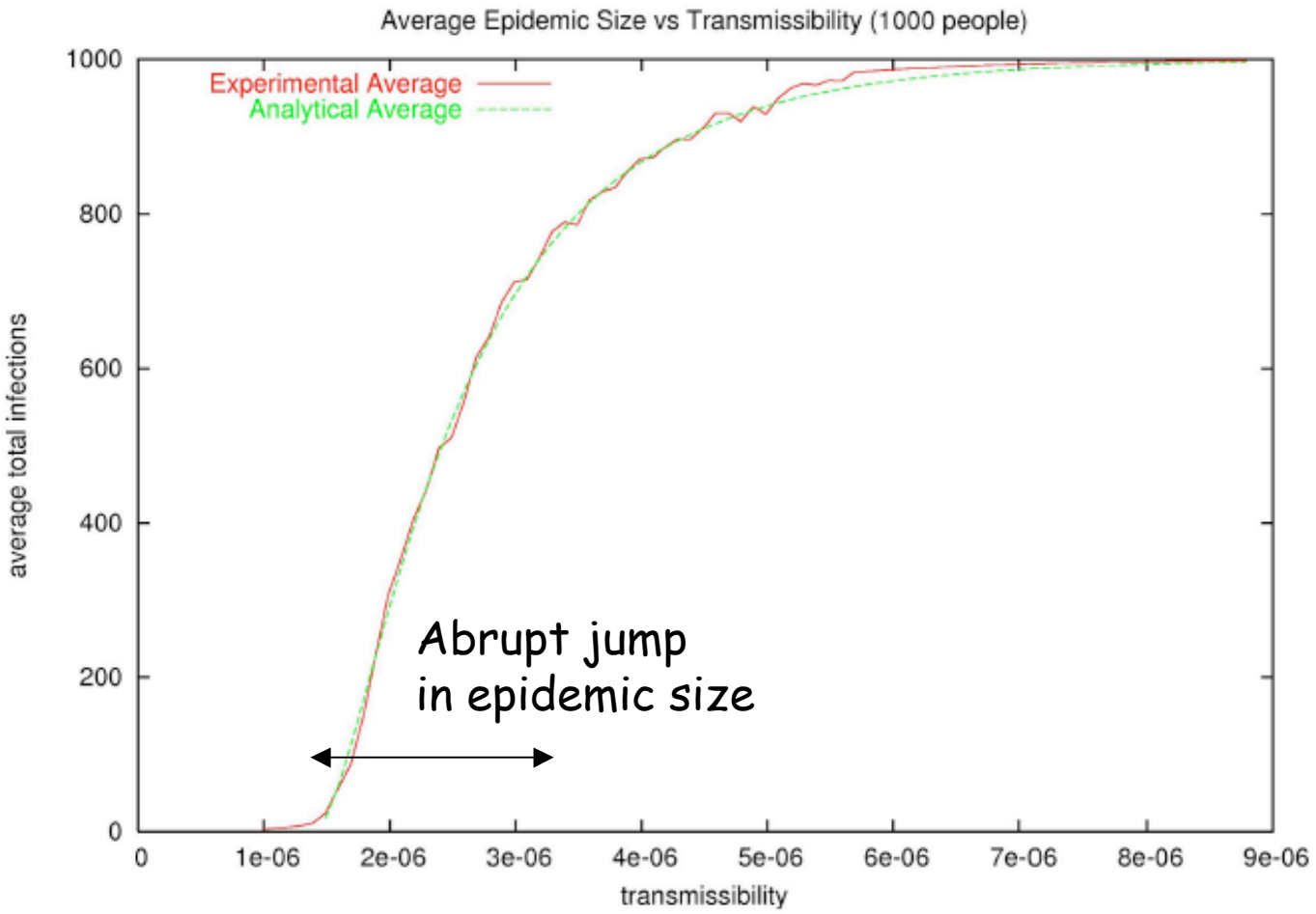
$P_S[R] = p(1,3)p(3,2)(1-p(1,2))^2(1-p(3,4))^2(1-p(2,4))^2$

Phase space for the SIR process





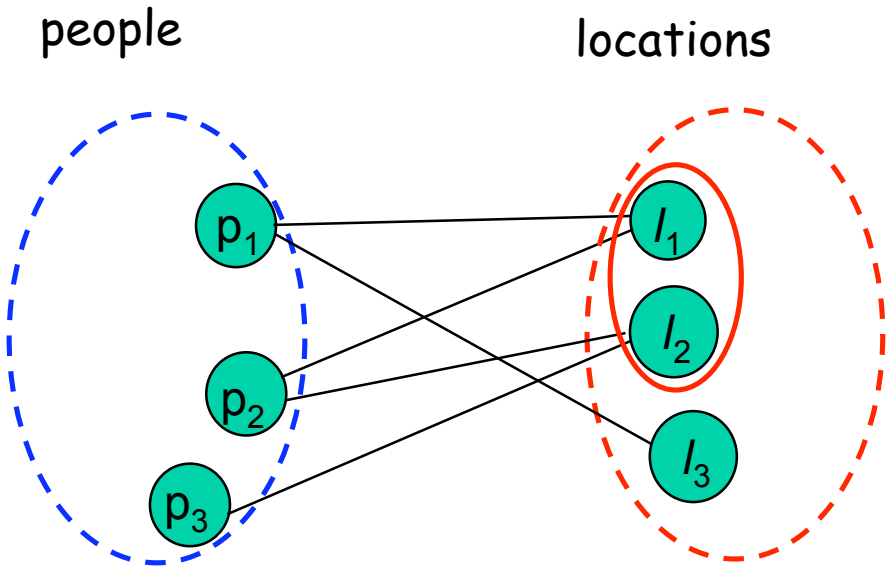
Threshold phenomena



Outline

- John Snow and cholera
- A combinatorial model for diseases
- A formulation for the sensor placement problem
- A formulation for the vaccination problem

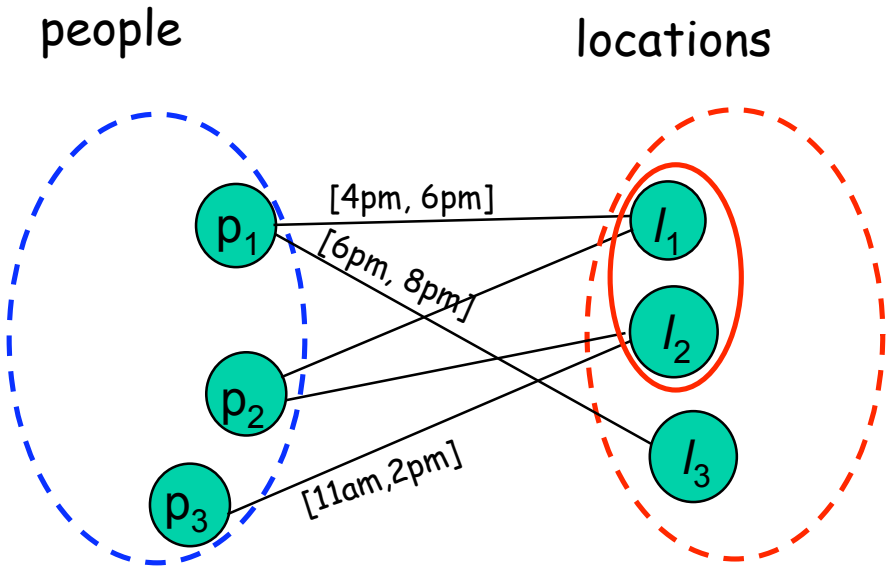
The Sensor Placement Problem



Dominating set problem: choose $L' \subseteq L$ s.t. $N(L') = P$

(1-ε)-Dominating set problem: choose $L' \subseteq L$ s.t. $|N(L')| \geq (1-\varepsilon)|P|$

Temporal Dominating Set



I(p, l): interval during which p visits l

The Temporal Dominating set problem: choose $L' \subseteq L$, and time interval $[t_1, t_2]$ for each $l \in L'$ s.t. $\forall\ p \in P, \exists\ l \in L'$ s.t. $I(p,l) \cap [t_1, t_2] \neq \emptyset$

Complexity of the Sensor Placement problem

- NP complete to solve optimally in general
- ß-approximation algorithm A: if

  #locations chosen by algo A ≤ ß·OPT
  - OPT: smallest #locations possible for the given instance
  - Guarantee should be true for every instance (worst case approximation guarantee)
  - Algo runs in polynomial time

- No approximation better than c · log n possible in general
- Greedy algo gives O(log n) approximation

Greedy Algorithm for Sensor Placement

- Initialize
  - S=Ø (locations chosen)
  - C=Ø (people covered)
- Repeat till C=P (all elements are covered)
  - Choose location v that covers the largest number of uncovered people
  - Add v to S
  - Add all newly covered people to C

Running time of Greedy algorithm

- Naive implementation: $O(m|L|)$ time
- Improved implementation: $O(m \log |L|)$ time
  - Using priority queues
  - Main operations
    - Find location v that covers the most number of uncovered people
    - Update coverage of all other locations after v is added to S: for each $p \in P$ covered by v, reduce coverage of all v' visited by p

Priority Queues: Basic Operations

- Store a set of elements, with key(v) for each v
- Find the element with smallest key and remove it
- Update the key of some element

- Store as a list
  - $O(1)$ time to insert/delete element
  - $O(n)$ to find minimum

- Store as a sorted array
  - $O(1)$ to find minimum
  - $O(\log n)$ time to find any element
  - $O(n)$ to delete/insert

- Use priority queues to implement Greedy?

Heaps

- Combine benefits of both lists and sorted arrays
- Stored in a balanced binary tree T
- Heap order: for each element v at node i of T, the element w at i's parent satisfies key(w) ≤ key(v)
- Assume N=maximum number of elements known in advance
- Store nodes of T in array
- node at position i has children in positions 2i and 2i+1
- parent(i) = $\lfloor i/2 \rfloor$

Elements in a heap

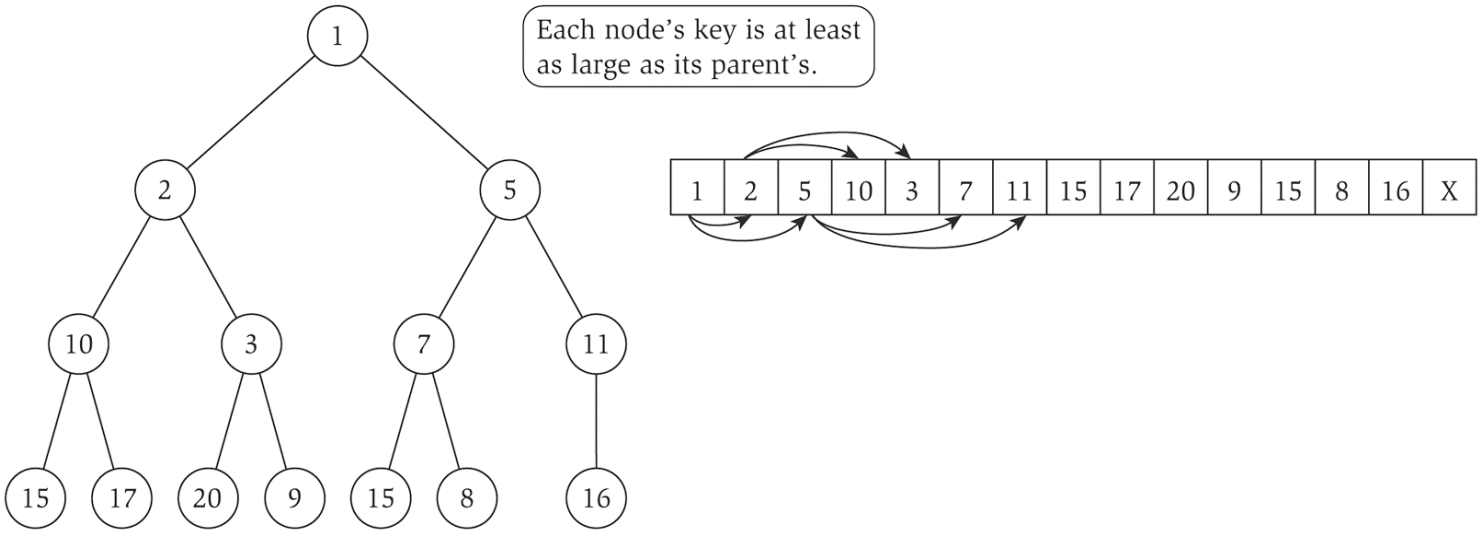


Figure 2.3 Values in a heap shown as a binary tree on the left, and represented as an array on the right. The arrows show the children for the top three nodes in the tree.

Inserting an element

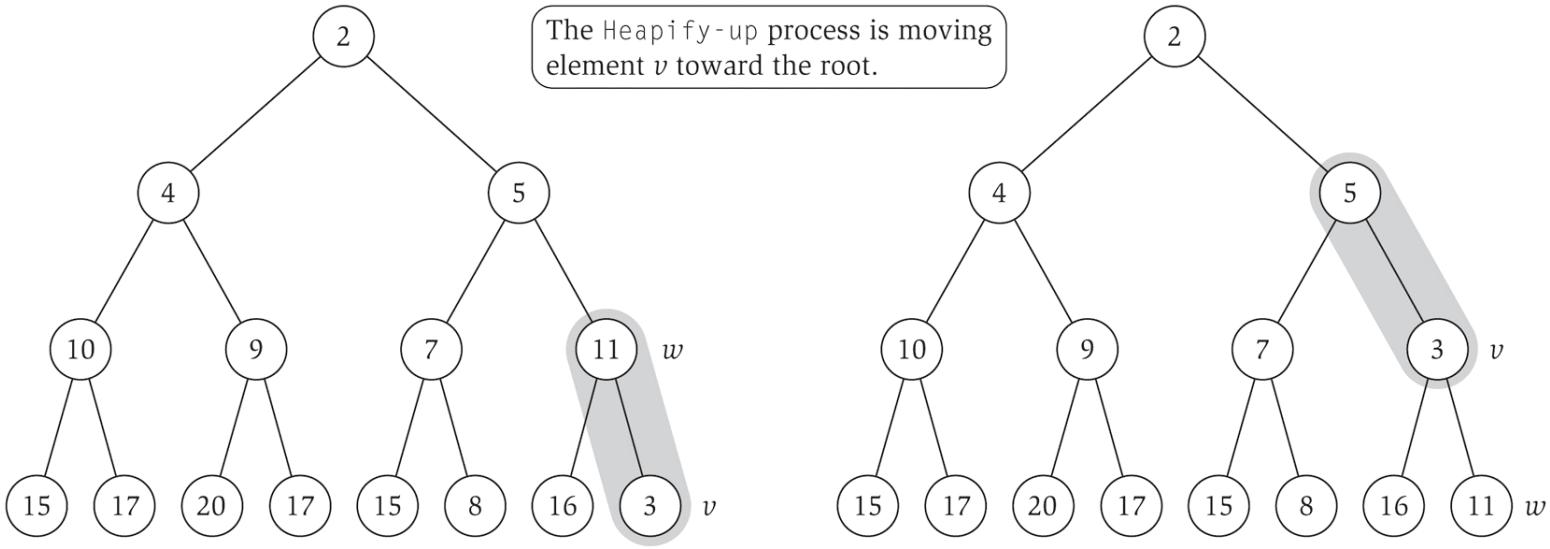


Figure 2.4 The Heapify-up process. Key 3 (at position 16) is too small (on the left). After swapping keys 3 and 11, the heap violation moves one step closer to the root of the tree (on the right).

Insert new element at position n+1 in array
Fix heap order by Heapify-up

Heapify-up

```
Heapify-up(H,i):
  If i > 1 then
    let j = parent(i) = ⌊i/2⌋
    If key[H[i]]<key[H[j]] then
      swap the array entries H[i] and H[j]
      Heapify-up(H,j)
    Endif
  Endif
```

Let v be the element in H at position i
Induction hypothesis: H is almost a heap with key of H[i] too small, i.e., there is a value a ≥ key(v) s.t. increasing key(v) to a would make H a heap

Base case: i=1 true
Consider i>1. Let v=H[i], j=parent(i), w=H[j]
After swapping v and w, either H is a heap or almost a heap with H[j] too small - true because setting a value of H[j] to key(w) ≥ key(H[j]) would make H a heap

O(log n) steps to run Heapify-up

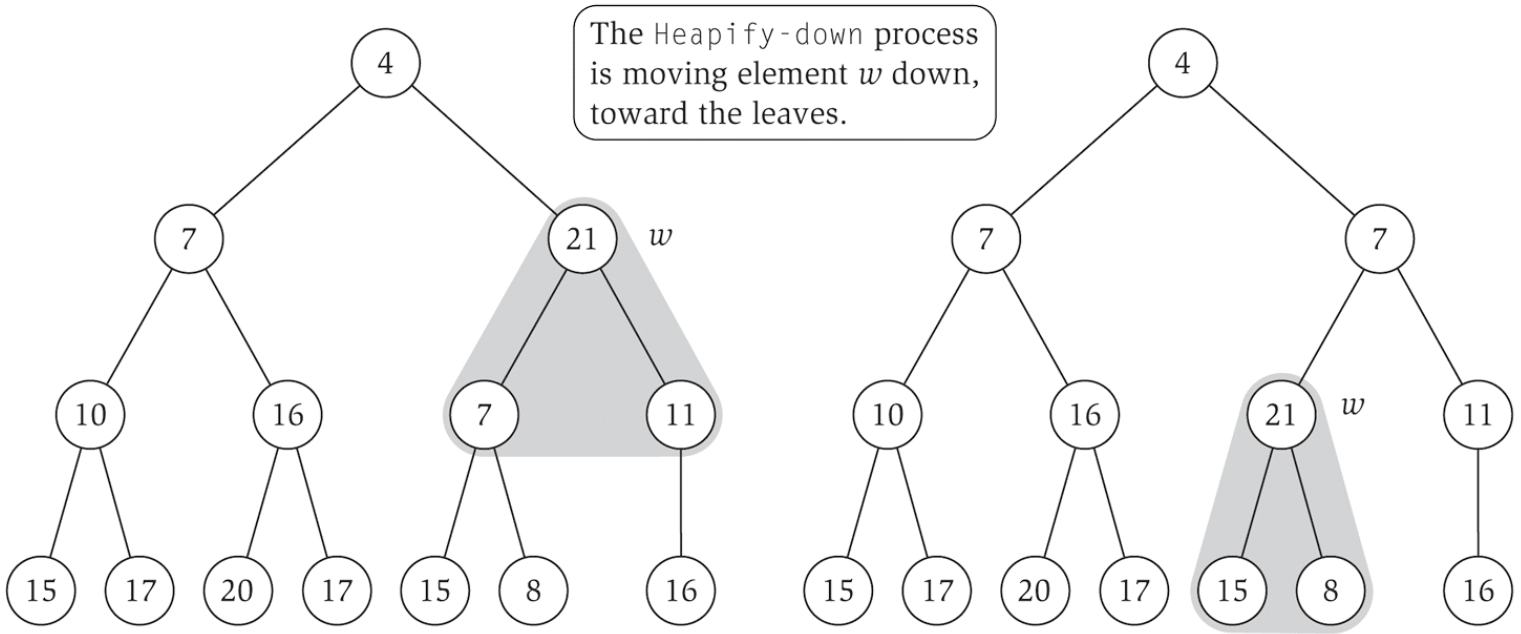


Figure 2.5 The `Heapify-down` process:. Key 21 (at position 3) is too big (on the left). After swapping keys 21 and 7, the heap violation moves one step closer to the bottom of the tree (on the right).

Heapify-down

```
Heapify-down(H,i):
  Let n = length(H)
  If 2i > n then
    Terminate with H unchanged
  Else if 2i < n then
    Let left = 2i, and right = 2i + 1
    Let j be the index that minimizes key[H[left]] and key[H[right]]
  Else if 2i = n then
    Let j = 2i
  Endif
  If key[H[j]] < key[H[i]] then
     swap the array entries H[i] and H[j]
     Heapify-down(H, j)
  Endif
```

Proof: Heapify-down

- Definition: H is almost a heap with the key of H[i] too big if there is a value a ≤ key(H[i]) such that lowering the key of H[i] to a will satisfy the heap property.
- Theorem: The Heapify-down leads to a valid heap.
- Proof: If element H[i] is a leaf, i.e., 2i>n, H is a heap.
- Induction Hypothesis: If H is almost a heap with the key of H[i] too big, then Heapify-down(H,i) is almost a heap with the key of H[j] too big, where H[j] is a child of H[i], if 2i<n.
- After swap: heap property may only be violated at position j.

- ⇒ Heapify-down leads to heap property in $O(\log n)$ steps

Approximation guarantee of Greedy

- Let Greedy pick the locations $v_1, v_2, \ldots, v_k$ (in this order)
- Let $S(v_i)$ be the set of people covered by v_i alone (not covered by locations $v_1, \ldots, v_{i-1}$)
- For each $p \in S(v_i)$ choose $f(p) = 1/|S(v_i)|$
- $\sum_p f(p) = k$ = cost of greedy
- For each location v in OPT
  - $\sum_{p \text{ in } S(v)} f(p) \leq \log_e n$

Theorem #sets chosen by Greedy $\leq (\log_e n)$ OPT

Sensor Placement in practice

- FastGreedy: choose large locations in non-increasing order of degrees whose sum of degrees is $(1-\varepsilon')|P|$
- FastGreedy works well in practice
  - 10% of the locations can dominate ~90% of people in Portland network
  - Very close to Greedy
  - Takes ~15 sec
- Temporal version hard to approximate within $\Omega(n^{\varepsilon})$

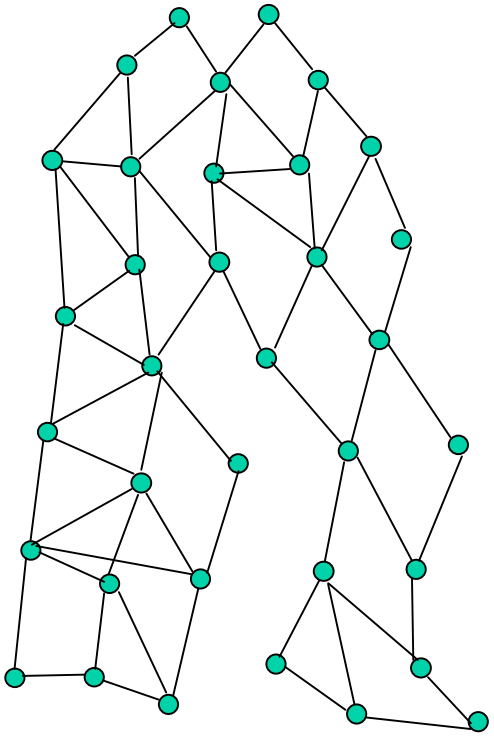
Outline

- John Snow and cholera
- A combinatorial model for diseases
- A formulation for the sensor placement problem
- A formulation for the vaccination problem

Policy problems: whom to vaccinate

Given

- a social network G(P,E)
- initial infected set A
- budget B on # vaccinations

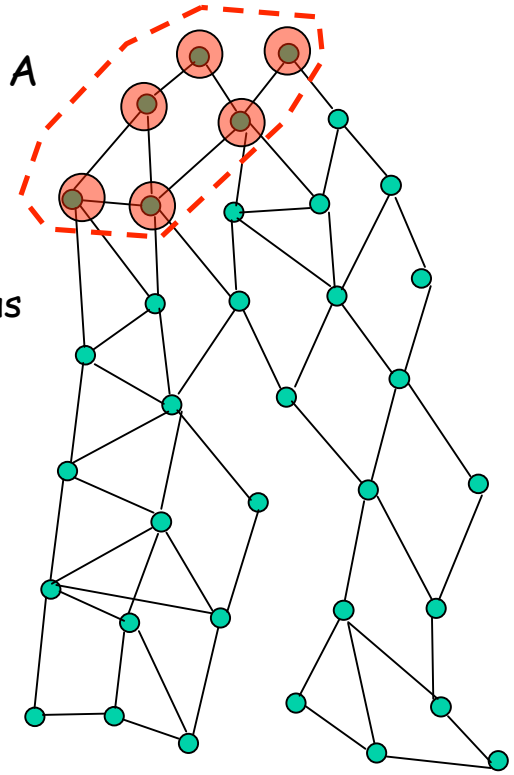


Policy problems: whom to vaccinate

Given
- a social network G(P,E)
- initial infected set A
- budget B on # vaccinations

Goal

choose $S \subseteq P$ to vaccinate so that $|S| \leq B$, and # nodes reachable from A in G[P\S] is minimized



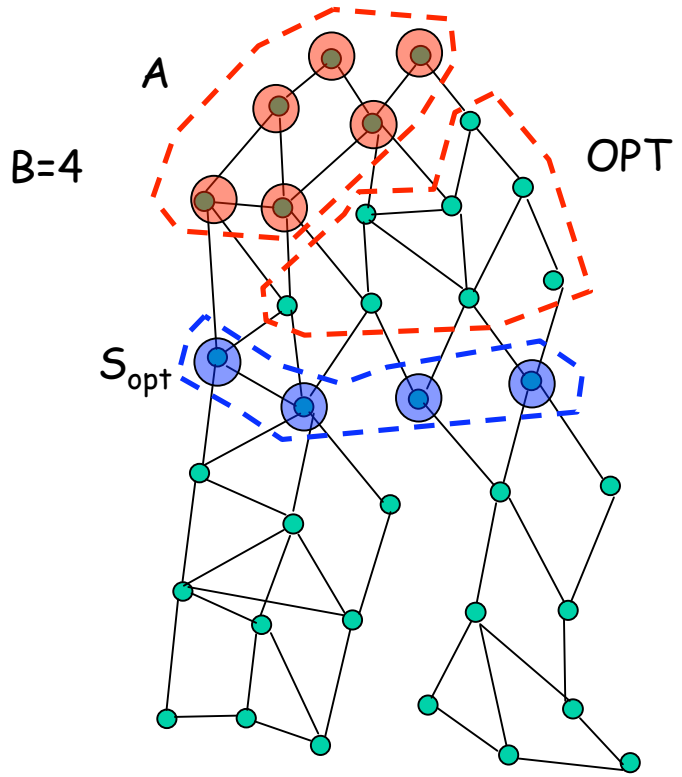
Assume: highly infectious disease
⇒ any node with a path to A gets infected

Policy problems: whom to vaccinate

Given
- a social network G(P,E)
- initial infected set A
- budget B on # vaccinations

Goal
choose $S \subseteq P$ to vaccinate so that $|S| \leq B$, and # nodes reachable from A in G[P\S] is minimized

Result
Bicriteria approximation: Vaccinate (1+ε)B nodes so that at most (1+1/ ε) OPT infected people[1,2]



[1]S. Eubank, V.S. Anil Kumar, M. Marathe, A. Srinivasan and N. Wang, AMS-DIMACS special volume, 2005
[2]A. Hyrapetyan, D. Kempe, M. Pal and Z. Svitkina, ESA 2005