Combinatorial optimization and graphical models

Submodular functions

Optimization - 10725
Carlos Guestrin
Carnegie Mellon University

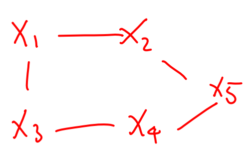
April 28th, 2008

Most probable explanation (MPE) in a Markov network

$\psi_i(x_i) = e^{f_i(x_i)}$

$\psi_{ij}(x_i,x_j) = e^{f_{ij}(x_i,x_j)}$

- Markov net:



$$P(X) = \frac{1}{Z}\prod_i \psi_i(x_i) \prod_{ij} \psi_{ij}(x_i,x_j)$$

$$= \frac{1}{Z} e^{\sum_i f_i(x_i) + \sum_{ij} f_{ij}(x_i,x_j)}$$

$$Z = \sum_x e^{\sum_i f_i(x_i) + \sum_{ij} f_{ij}(x_i,x_j)}$$

- Most probable explanation:

$$\arg\max_x P(x) = \arg\max_x \log P(x) = \arg\max_x \underbrace{\sum_i f_i(x_i) + \sum_{ij} f_{ij}(x_i,x_j)}_{} - \cancel{\log Z}$$

- In general, NP-complete problem, and hard to approximate

MPE for attractive MNs – 2 classes

$|Val(X_i)| = 2$

- Attractive MN:
  - E.g., image classification

$f_i(x_i) \leftarrow$ anything — local measurement

$\alpha_{i0} \geq 0$, $\alpha_{i1} \geq 0$ — w.l.o.g. add most negative to all α's

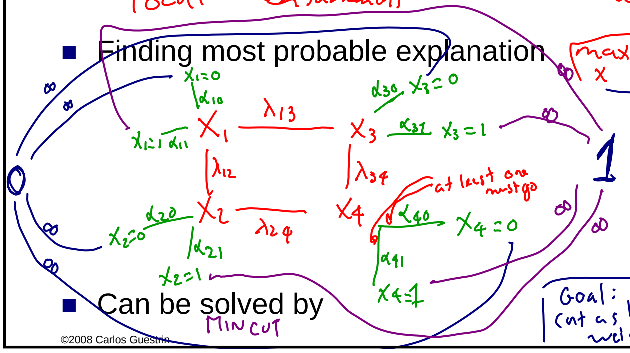
$$f_{ij}(x_i, x_j) = \begin{cases} \lambda_{ij}, & x_i = x_j \\ 0, & \text{otherwise} \end{cases}$$

smoothness "constraint"

$\lambda_{ij} \geq 0$

- Finding most probable explanation

$$\max_x \sum_i f_i(x_i) + \sum_{ij} f_{ij}(x_i, x_j)$$



OPT $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$

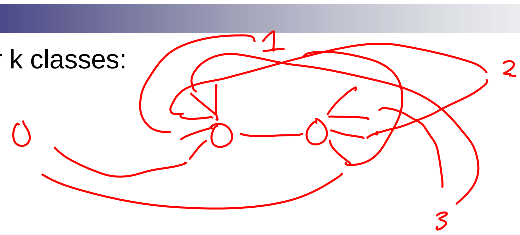
Val OPT: $\lambda_{12} + \lambda_{34} + \alpha_{11} + \alpha_{21} + \alpha_{30} + \alpha_{40}$

Goal: cut as little edge weight as possible

- Can be solved by MIN CUT

MPE, Attractive MNs, k classes

- MPE for k classes:

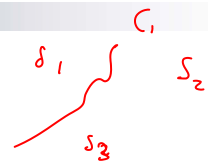


- Multiway cut:
  - Graph G, edge weights w_{ij}
  - Finding minimum cut, separate $s_1, \ldots, s_k$

- Multiway cut problem is NP-complete

There is a $\frac{3}{2}$ approx alg, LP duality

Multiway cut – combinatorial algorithm



- Very simple alg:
  - For each i=1…k
    - Find cut C_i that separates s_i from rest
  - Discard $\arg\max_i w(C_i)$, return union of rest

$C_1 \leftarrow w(C_1) = 5$

$C_2 \leftarrow w(C_2) = 8$

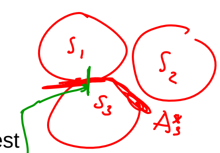
$C_3 \leftarrow w(C_3) = 6$

return $C_1 \cup C_3$

$w(C) = \sum_{i,j \in C} w_{ij}$

- Algorithm achieves 2-2/k approximation
  - OPT cut A* separates graph into k components
    - No advantage in more than k

$w_{ij} \geq 0$



  - From A* form $A^*_1,\ldots,A^*_k$, where A^*_i separates s_i from rest

edge appears in A_1^* & A_3^*

  - Each edge in A* appears in exactly 2 A_i^*
    - Thus

$2w(A^*) = \sum_i w(A_i^*)$

$w(C_i) \leq w(A_i^*)$

Multiway cut proof

- Thus, for OPT cut A* we have that:

$$2\,w(A^*) = \sum_i w(A_i^*)$$

- Each A^*_i separates s_i from rest, thus

$$w(C_i) \leq w(A_i^*)$$

but C_i is OPT cut of s_i from rest

- But, can do better, because

throw away $\max_i w(C_i) \geq \frac{1}{k} \sum_i w(C_i)$

$$\sum_i w(C_i) - \max_i w(C_i) \leq \left(1-\frac{1}{k}\right) \sum_i w(C_i)$$

$$\leq \left(1-\frac{1}{k}\right) \sum_i w(A_i^*) = 2\left(1-\frac{1}{k}\right) w(A^*)$$

General solutions to combinatorial problems

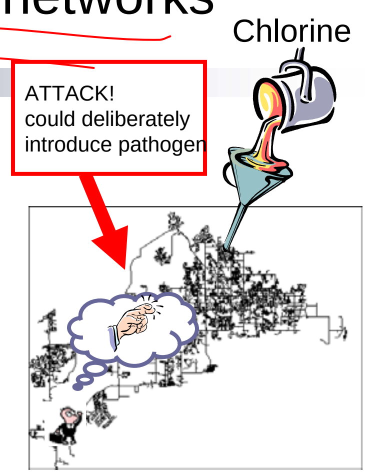
- Nonlinear optimization
  - Extremely general
  - NP-Hard to solve
- Covex problems:
  - Special problem structure
  - General efficient solution algorithms
  - Applies to many continuous problems
- Combinatorial optimization
  - Integer programming
    - Very general, but NP-hard
  - Convex relaxations
    - Approximation guarantees for many problems
    - Usually solution is problem specific
      - Though there are general principles
- Is there general problem structure in combinatorial problems?
  - Analogously to convexity?
    - Recognize structure and use general algorithms?

$\theta f(x) + (1-\theta) f(y) \geq f(\theta x + (1-\theta) y)$

e.g., totally unimodular matrices

Water distribution networks

- Water distribution in a city → very complex system
- Pathogens in water can affect thousands (or millions) of people
- Currently: Add chlorine to the source and hope for the best

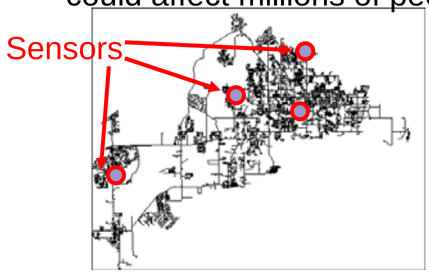


Simulator from EPA

Monitoring water networks

[Krause, Leskovec, G., Faloutsos, VanBriesen '08]

- Contamination of drinking water could affect millions of people



Simulator from EPA

$A \subseteq V$

Hach Sensor

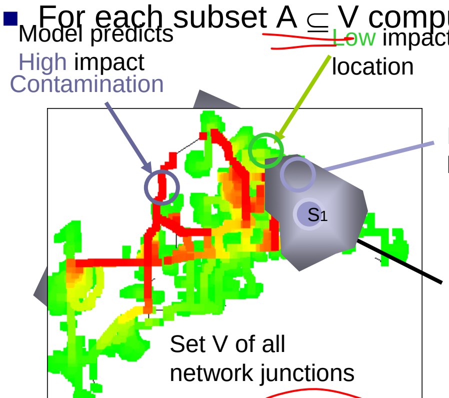
~$14K

- Place sensors to detect contaminations
- "Battle of the Water Sensor Networks" competition

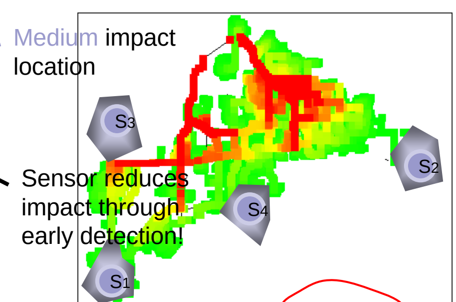
Where should we place sensors to detect contaminations quickly ?

Model-based sensing

- Model predicts impact of contaminations
  - For water networks: Water flow simulator from EPA
  - For lake monitoring: Learn probabilistic models from data (later)
- For each subset $A \subseteq V$ compute "sensing quality" F(A)



High sensing quality F(A) = 0.9



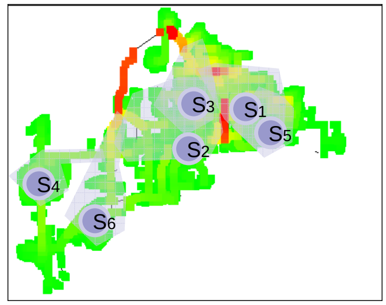
Low sensing quality F(A)=0.01

Sensor placement

- Given: finite set V of locations, sensing quality F
- Want: $A^* \subseteq V$ such that

$$\mathcal{A}^* = \operatorname*{argmax}_{|\mathcal{A}|\leq k} F(\mathcal{A})$$

Typically NP-hard!



Greedy algorithm:

Start with A = Ø

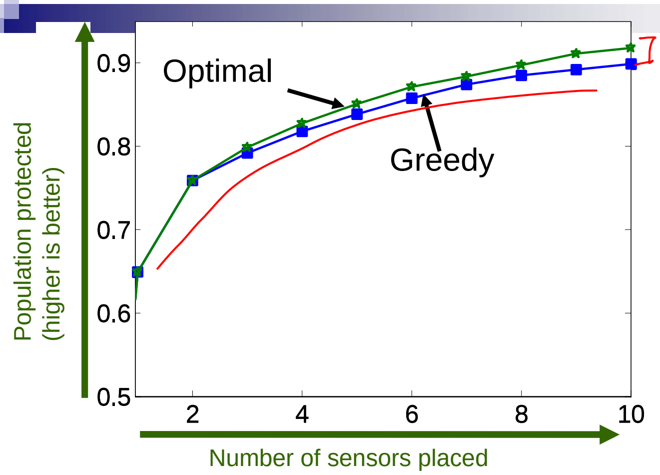
For i = 1 to k

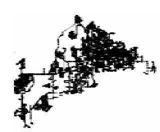
$s^* := \operatorname{argmax}_s F(A \cup \{s\})$

$A := A \cup \{s^*\}$

How well can this simple heuristic do?

Performance of greedy algorithm





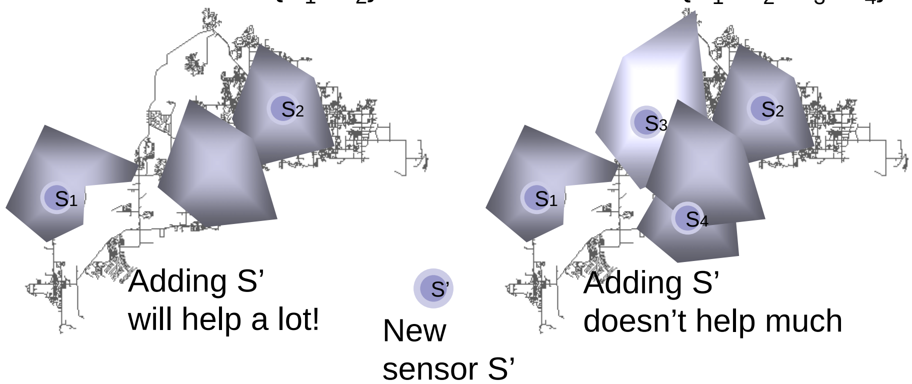
Small subset of Water networks data

Greedy score empirically close to optimal. Why?

Key property: Diminishing returns

Submodularity

Placement A = $\{S_1, S_2\}$ Placement B = $\{S_1, S_2, S_3, S_4\}$



Theorem [Krause, Leskovec, G., Faloutsos, VanBriesen '08]:

Sensing quality F(A) in water networks is submodular!

Submodularity

supermodular $F(A \cup \{x\}) - F(A) \leq F(B \cup \{x\}) - F(B)$

$A \subset B$

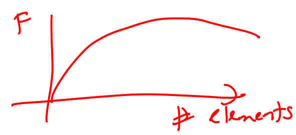
modular $F(A \cup \{x\}) - F(A) = F(B \cup \{x\}) - F(B)$

$\Leftrightarrow F(A) = \sum_{A_i \in A} F(A_i)$

- Formalizes notion of diminishing returns

$\forall A, B \quad F(A \cup \{x\}) - F(A) \geq F(B \cup \{x\}) - F(B)$

$A \subseteq B$



- Equivalent definition:

$\forall A, B \quad F(A) + F(B) \geq F(A \cup B) + F(A \cap B)$

What do we get from submodularity

max $f(x)$
x
e.g. $x \in$
Hard

- Submodularity is a general property of set functions
- Submodular function can be minimized in polynomial time!

$$\min_{A \subseteq V} F(A)$$

F is submodular
polytime

- But our problem is

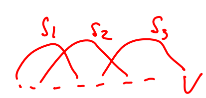
water problem

$$\max_{A,\ |A| \le k} F(A)$$

F is submodular

Another example…

- Maximum cover
  - Ground elements $v \in V$
  - Set of sets $S_i \subseteq V$
  - Pick k sets, maximize number of covered elements



$$F(A) = \sum_v I(v \text{ is } \bigcup_{i \in A} S_i)$$

A is a set of sets

$A = \{S_1, S_2, \ldots S_k\}$

$$F(A \cup \{x\}) - F(A) \geq F(B \cup \{x\}) - F(B)$$
$$A \subseteq B$$

consider element v
- covered A
- covered by x, but not B or A
- covered B, but not A

Maximizing submodular functions – cardinality constraint

- Given
  - Submodular function $F(A)$
  - Normalized $F(\emptyset) = 0$
  - Non-decreasing $F(A) \leq F(B)$ $A \subseteq B$

e.g.) max cover

- Greedy algorithm guarantees

$$F(A_{Greedy}) \geq \underbrace{\left(1 - \frac{1}{e}\right)}_{63\%} F(A_{OPT})$$

$$\max_{A,\ |A| \leq k} F(A)$$

- Can you get better algorithm?

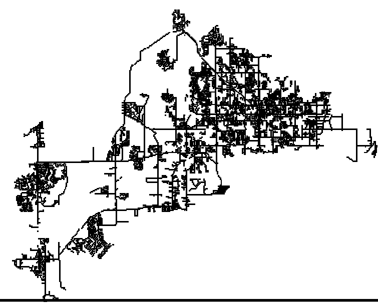
NO, unless P=NP, if you could solve max-cover better than $(1-1/e)$ then P=NP

Online bounds

- Submodularity provides bounds on the quality of the solution A obtained by any algorithm
  - For normalized, non-decreasing functions
- Advantage of adding elements to A:
- Bound on the quality of any set A:
- Tighter bound:

Battle of the Water Sensor Networks Competition

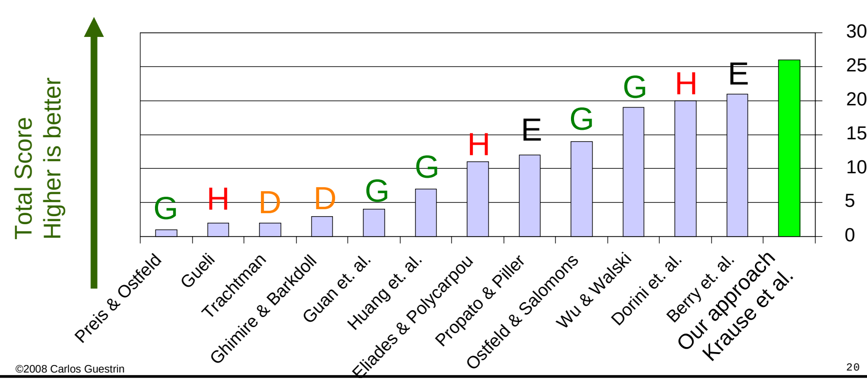
- Real metropolitan area network (12,527 nodes)
- Water flow simulator provided by EPA
- 3.6 million contamination events
- Multiple objectives: Detection time, affected population, …
- Place sensors that detect well "on average"



BWSN Competition results

- 13 participants
- Performance measured in 30 different criteria

G: Genetic algorithm
D: Domain knowledge
H: Other heuristic
E: "Exact" method (MIP)

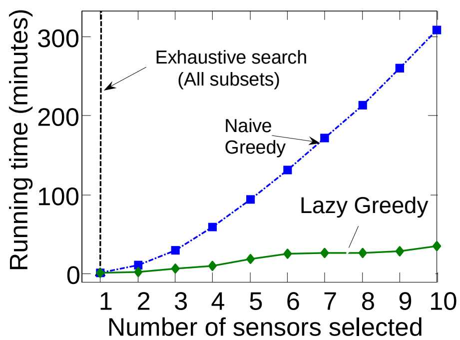


What was the trick?

Simulated all 3.6M contaminations on 2 weeks / 40 processors

152 GB data on disk, 16 GB in main memory (compressed)

➔ Very accurate sensing quality ☺ Very slow evaluation of F(A) ☹



30 hours/20 sensors

6 weeks for all
30 settings ☹

ubmodularity
to the rescue:

Using "lazy evaluations":
1 hour/20 sensors

Done after 2 days! ☺

Lazy evaluations

- Naïve implementation of greedy:
- Advantage of an element never increases:
  - Advantage:
  - What if you already picked a larger set:
    - Set after picking i elements: A_i
- Lazy evaluations:
  - Keep a priority queue over elements:
    - Initialize with advantage of each element
  - Pick element on top, recompute priority
  - If element remains on top

Other maximization settings

- Non-monotone submodular functions
- Non-unit costs
- Complex constraints
  - Paths
  - Spanning trees
- Worst-case optimization