

Lecture 12, Nov 28

Linear and Integer Programming

A Linear Program

- A “linear program” is an optimization problem consisting of:
 - an “objective function”:
$$c_1 x_1 + c_2 x_2 + \dots + c_n x_n$$
$$c_i$$
 constants, x_i variables (all over real numbers).
 - m linear “constraints”: i.e., of the form
$$a_{\{i,1\}} x_1 + a_{\{i,2\}} x_2 + \dots + a_{\{i,n\}} x_n \leq / = / \geq b_i$$
for $i = 1, 2, \dots, m$.

Linear Programming: Goal

- The goal is to find real values of x_i 's that maximize/minimize the objective function and satisfy all constraints.
- This problem is solvable in polynomial time.
- Integer programming: the same problem, but all the constraints and the variables must be integer.
- Is integer programming harder than LP?

Is integer programming harder than the general LP?

- Yes.
- IP is NP-hard.
- We'll see a few examples of NP-hard problems that can be solved with integer programming.

Formulating Shortest Path as an LP

- Goal: Given a weighted graph G with positive weights, and two vertices s and t , find the length of the shortest path from s to t .

Shortest Path as an LP - solution

- Construct linear program with variables d_v for each v in V :
- Maximize: d_t
- Subject to: $d_v \leq d_u + w(u,v)$ for each (u,v) in E
- $d_s = 0$
- Why does it work?

Why does it work?

- First of all, the actual distances $d_v = d(s,v)$ satisfy all the conditions, hence $d_t(\max) \geq d(s,t)$.
- On the other hand if $P=s-v_1-v_2-\dots-v_k-t$ is a shortest path from s to t , then $d_t \leq d_{v_k} + w(v_k,t) \leq d_{v_{k-1}} + w(v_{k-1},v_k) + w(v_k,t) \leq \dots \leq d_s + w(s,v_1) + \dots + w(v_{k-1},v_k) + w(v_k,t) = 0 + \text{length}(P)$.
- Hence $d_t \leq \text{length}(P)$ for all paths P from s to t , and $d_t \leq d(s,t)$.

Formulating Network Flows as LP

- Given network $N=(V,E)$ with capacities $c(e)$ for e in E ,
- Construct linear program with variables $f_{(u,v)}$ for each edge (u,v) in E .

The constraints:

- The constraints are the properties of

(i) (*Capacity conditions.*) For each $e \in E$, $0 \leq f(e) \leq c_e$.

(ii) (*Conservation conditions.*) For each node v other than s and t , we have

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

- All are linear constraints.

Goal:

- Maximize the flow

$$|f| = \sum_{v \in V} f(s,v) = \sum_{v \in V} f(v,t)$$

subject to the constraints above.

Vertex Cover

- Problem statement:
- Input: Undirected graph $G=(V,E)$.
- Output: Subset of vertices C that "covers" every edge (i.e., each edge has at least one endpoint in C), with minimum size.
- NP-hard in general.

IP Solution for Vertex Cover

- Use variable x_i for each vertex v_i in V
- Minimize: $x_1 + x_2 + \dots + x_n$
- Subject to: $x_i + x_j \geq 1$ for all (v_i, v_j) in E ;
- x_i in $\{0,1\}$ for all v_i in V : $0 \leq x_i \leq 1$.
- Integer program is completely equivalent to original problem, including NP completeness (so no polytime algorithm).

Linear relaxation:

- Remove restriction of x_i to integer values:
- Minimize: $x_1 + x_2 + \dots + x_n$
- subject to: $x_i + x_j \geq 1$ for all (v_i, v_j) in E ;
- $0 \leq x_i \leq 1$ for all v_i in V .
- Solution can be found in polytime, but may include fractional values of x_i 's.
- Does not immediately solve the original problem.

Example:

- $G=(V,E)$ where $V=\{1,2,3\}$,
 $E=\{(1,2),(2,3),(1,3)\}$ (a triangle) becomes
- minimize: $x_1 + x_2 + x_3$
- subject to: $x_1 + x_2 \geq 1$;
 $x_2 + x_3 \geq 1$;
 $x_1 + x_3 \geq 1$;
 $0 \leq x_1, x_2, x_3 \leq 1$.
- With solution $x_1 = x_2 = x_3 = 1/2$.

Rounding:

- Compute optimal solution to linear program: x'_1, x'_2, \dots, x'_n .
- We create a cover by rounding the fractional solution.
- Create cover as follows:
for each v_i in V , put v_i in C iff $x'_i \geq 1/2$.
- C is a cover because constraint $x_i + x_j \geq 1$ guarantees at least one of $x'_i, x'_j \geq 1/2$ for each edge (v_i, v_j) .
- Gives a 2-approximation to the optimal.

Why is C a 2-approximation?

- Let C^* be the optimal vertex cover.
- Let $x_i^*=1$ iff the vertex v_i is covered in C^* .
- Because C^* is a VC, the x_i^* satisfy the constraints of the linear program, and so the optimal solution (x'_1, \dots, x'_n) is at least as good as (x_1^*, \dots, x_n^*) . So
$$x'_1 + \dots + x'_n \leq |C^*|.$$
- Let (x_1, \dots, x_n) be the solution obtained by rounding the optimal fractional solution (x'_1, \dots, x'_n) .

Why is C a 2-approximation? – cont'd

- For each i , $x_i \leq 2 x'_i$ (check!).
- So,
$$|C| = x_1 + \dots + x_n \leq 2(x'_1 + \dots + x'_n) \leq 2|C^*|.$$
- We see that the optimal solution is at most two times smaller than C .

Weighted vertex cover:

- Section 11.6.
- Now each vertex v_i has a weight w_i , and we need to minimize the total weight of the cover.
- The same approach still gives a 2-approximation of the optimum.
- What are the “worst” examples for the approximation?

A simple 2-approximation for (unweighted) vertex cover.

- Repeatedly pick an edge and put both endpoints in C , then remove all edges incident on the two endpoints, until no edge remains.
- $|C| \leq 2 * OPT$ (size of smallest cover).
- Suppose C contains the endpoints of the edges e_1, e_2, \dots, e_k – thus $|C|=2k$.
- By the construction, these edges are disjoint and every VC must include at least one endpoint of each of the k edges. Hence $|C_{OPT}| \geq k = |C|/2$, and C is a 2-approximation.

Load Balancing Problem

- Given m machines and n jobs.
- Each job j has a processing time of t_j .
- Want to assign the jobs to the machines so that all the machines finish as soon as possible.
- Intuitively, this is achieved by “balancing” the work between the machines.

Parameters

- The machines M_1, M_2, \dots, M_m .
- The set of jobs assigned to M_i is $A(i)$.
- The T_i time M_i needs:
this is the load on M_i $T_i = \sum_{j \in A(i)} t_j$
- Want to minimize makespan: $T = \max_i T_i$
- What if $m > n$?

Simple greedy algorithm

```
Greedy-Balance
Start with no jobs assigned
Set  $T_i = 0$  and  $A(i) = \emptyset$  for all machines  $M_i$ 
For  $j = 1, \dots, n$ 
  Let  $M_i$  be a machine that achieves the minimum  $\min_k T_k$ 
  Assign job  $j$  to machine  $M_i$ 
  Set  $A(i) \leftarrow A(i) \cup \{j\}$ 
  Set  $T_i \leftarrow T_i + t_j$ 
EndFor
```

Simple greedy algorithm – cont'd

- Why doesn't it always work?
- OK, it doesn't give the optimal solution, but maybe the solution it outputs is not much worse than the optimal one.
- Denote the output of the algorithm by T and the optimal solution by T^* .

Simple greedy algorithm – cont'd

- OK, it doesn't give the optimal solution, but maybe the solution it outputs is not much worse than the optimal one.
- To find out, we try to bound T^* from below.
- If T^* is not too small, then T is not much worse than T^* .

To be continued...