ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

M.Sc. Program in Computer Science
Department of Informatics

Design and Analysis of Algorithms

Linear and Integer Programming

Flows, Matching, Vertex Cover, Set Cover

Vangelis Markakis

markakis@gmail.com

Linear Programming

Linear Programming

- Nothing to do with programming!
- A particular way of formulating certain optimization problems with linear constraints
- One of the most useful tools in Algorithms and Operations Research
- Extremely useful also in the design of approximation algorithms

Linear Programming

Example:

- A farmer possesses a land of 10 km^2
- He wants to plant the land with wheat, or barley or a combination of them
- The farmer has a limited amount of fertilizer, say 16 kgs
- And a limited amount of pesticide, say 18 kgs
- Each square km of wheat requires 1kg of fertilizer and 2 kgs of pesticide
- Each square km of barley requires 2kg of fertilizer and 1.2 kgs of pesticide
- Revenue to the farmer: 3 (thousand $) from each square km of wheat and 4 (thousand $) from each square km of barley
- Find out what the farmer should do

Linear Programming

Formulation as a linear program:

- Variables x_1, x_2
- x_1: number of square km for wheat
- Similarly, x_2 for barley
- Area constraint: $x_1 + x_2 \leq 10$
- Constraint for fertilizer: $x_1 + 2x_2 \leq 16$
- Constraint for pesticide: $2x_1 + 1.2x_2 \leq 18$
- Nonnegativity constraints: $x_1 \geq 0$, $x_2 \geq 0$ (cannot plant an area with negative surface)
- Objective function: maximize $3x_1 + 4x_2$
- Observe that: all constraints are linear, objective function also linear

Linear Programming

Usual writing style:

$$\max\ 3x_1 + 4x_2$$ ← Objective function

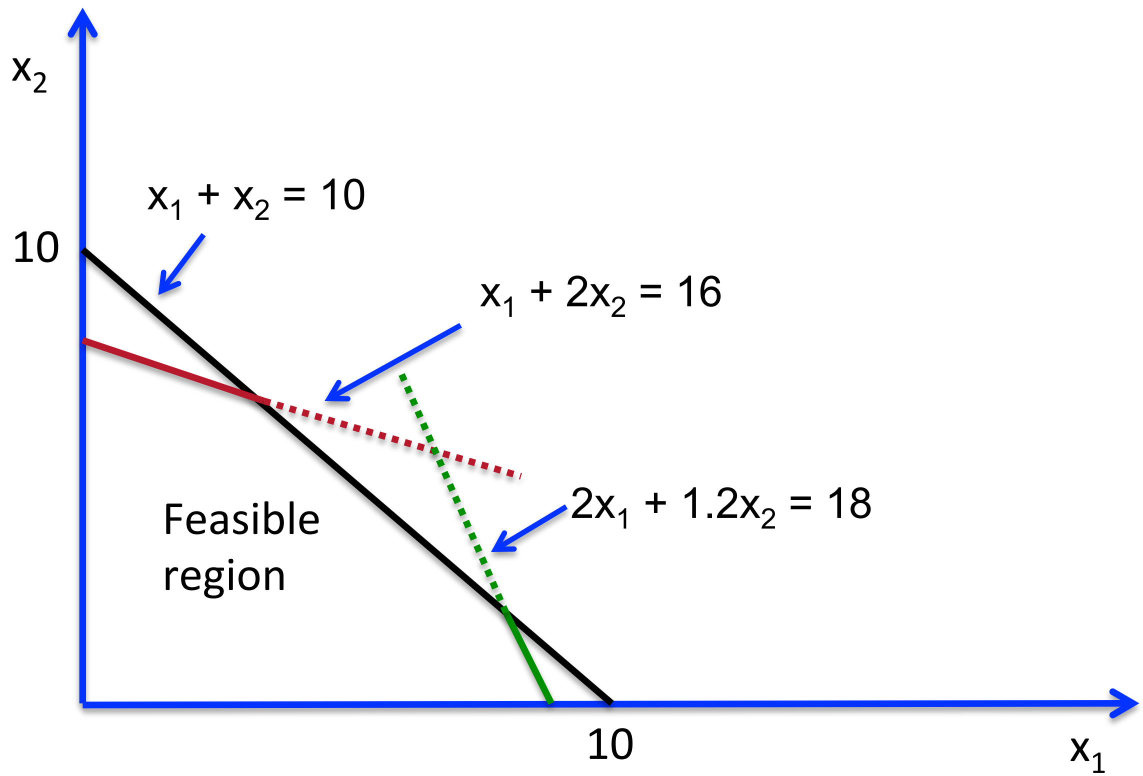
$$
\begin{aligned}
s.t.\quad & x_1 + x_2 \le 10 \\
& x_1 + 2x_2 \le 16 \\
& 2x_1 + 1.2x_2 \le 18 \\
& x_1, x_2 \ge 0
\end{aligned}
$$

} constraints

- It can be either a minimization or a maximization problem
- Feasible space (or region): the set of all pairs (x_1, x_2) that satisfy the constraints
- In the example: the feasible region is a subset of R^2
- It is always a polyhedron in R^n, where n = number of variables

Linear Programming

Geometrically:



Linear Programming

More succinct notation:

max. $c^T x$

s.t.

$Ax \le b$

$x \ge 0$

$$where\ c = \begin{pmatrix} 3 \\ 4 \end{pmatrix}, x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, b = \begin{pmatrix} 10 \\ 16 \\ 18 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 2 & 1.2 \end{pmatrix}$$

- We can also add slack variables to bring the constraints to the form

  $Ax = b, x \ge 0$
- Other problems may also not have the non-negativity constraints
- For solving purposes, these issues do not make a difference

Linear Programming

Complexity of linear programs:

- Believed to be NP-complete for quite some time
- In practice: run simplex and/or its variants
- Works extremely well on average, but it has worst case exponential time
- [Khachiyan '81]: the ellipsoid algorithm: the first polynomial time algorithm, very impractical though
- [Karmarkar '84]: a more efficient algorithm, forms the basis of today's interior point methods
- All you need to know about linear programs for this course: they can be solved efficiently both in theory and in practice!

Linear Programming

We will see 2 quick applications of LP

1. Flows in networks

2. Matching in bipartite graphs

Flows in Networks

(informal) problem statement:

Suppose we want to transport some quantity of a good within a given network, from some source to a destination

The good can be

- Oil to be transported through a network of oil pipes
- Information through a computer network
- Etc

Constraints: each edge in the network has a *capacity*, i.e., the maximum quantity it can carry

- oil pipes have a volume capacity
- A link in a computer network has limits on its bandwidth

Goal: find a way to route the good through the network so as to maximize the total quantity shipped

Flows in Networks

More formally:

Consider a graph $G = (V, E)$, with a source node $s \in V$, and a sink node $t \in V$

Capacity constraints: for every edge $e \in E$, there is a capacity c_e

A feasible flow is an assignment of a flow f_e to every edge so that

1. $f_e \leq c_e$
2. For every node other than source and sink:
   incoming flow = outgoing flow (preservation of flow)

Goal: find a feasible flow so as to maximize the total amount of flow coming out of s (or equivalently going into t)

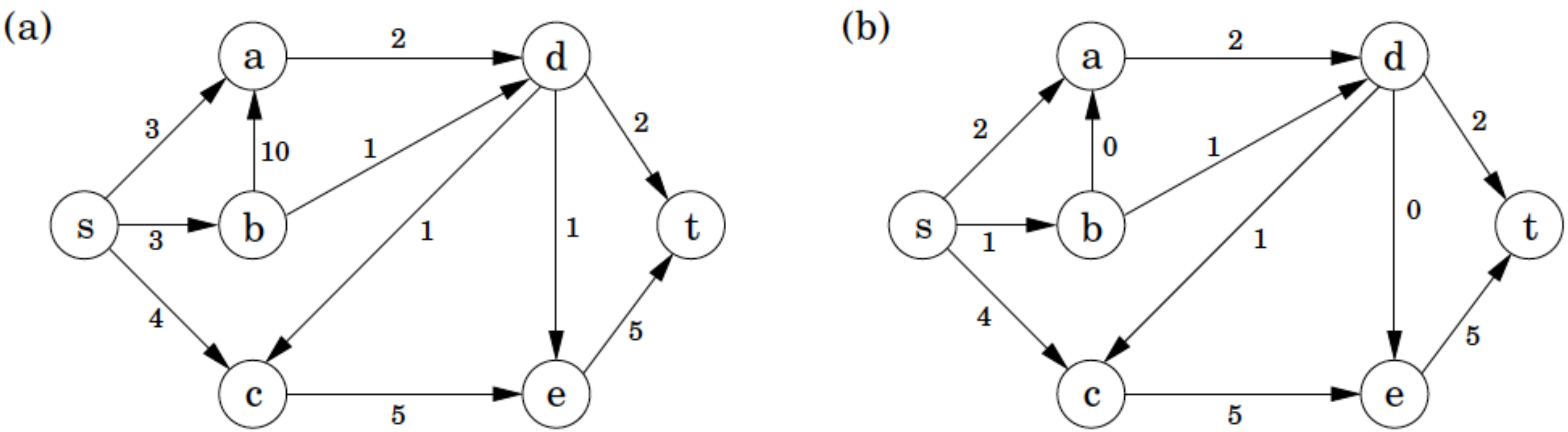
Flow going out of s: $\sum_{(s,u)\in E} f_{su}$

By preservation of flow this equals: $\sum_{(u,t)\in E} f_{ut}$

Flows in Networks

Example:

- Figure (a): network with capacities
- Figure (b): a feasible flow
- In fact, the flow in (b) is optimal (7 units)



Flows in Networks

Finding a max flow via Linear Programming:

- Suppose we use a variable f_{uv} for the flow carried by each edge
- Then, the objective function and all the constraints are linear

Objective function: $\sum_{(s,u)\in E} f_{su}$

Constraints

1. Capacity constraints: $f_{uv} \le c_{uv}$, for every $(u, v) \in E$
2. Non-negativity constraints: $f_{uv} \ge 0$, for every $(u, v) \in E$
3. Flow preservation: for every node $u \ne s, t$:

$$\sum_{(w,u)\in E} f_{wu} = \sum_{(u,v)\in E} f_{uv}$$

Flows in Networks

In the example of Figure (a):

max $f_{sa} + f_{sb} + f_{sc}$

s.t.

11 capacity constraints

11 non-negativity constraints

5 flow preservation constraints

27 constraints in total

Solving this => max flow = 7

Note: There are more efficient algorithms for solving max flow (not covered here)

- $O(|V|\ |E|^2)$ [Edmonds, Karp '72]
- $O(|V|^2\ |E|)$ [Goldberg '87]
- $O(|V|\ |E|\ \log(|V|^2/|E|))$ [Goldberg, Tarjan '86]

Flows in Networks

Certificates of optimality:

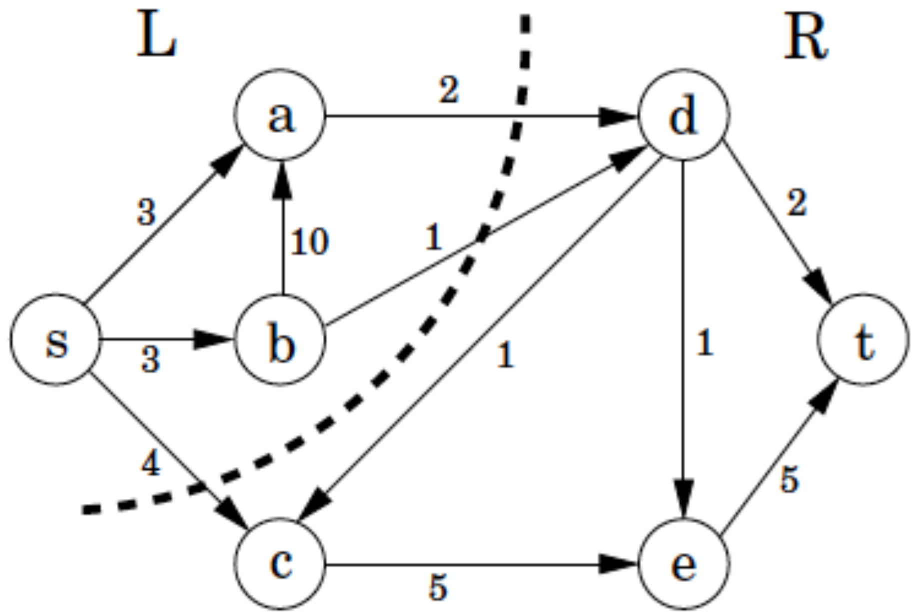
Suppose we have not solved the LP, but we have identified a feasible flow

Can we convince ourselves if it is optimal or not?

Definition: Given a graph G = (V, E), an s-t cut is a partition of the vertices into 2 sets, say L, R, such that $s \in L$, $t \in R$

Capacity of an s-t cut: sum of capacities of edges crossing the cut in the direction from L to R

Flows in Networks



capacity of cut = 7

Clearly:

max flow ≤ capacity of any s-t cut

(cannot send more flow to t than the capacity of the cut)

Hence:

max flow ≤ capacity of minimum s-t cut

Flows in Networks

In fact we have equality:

The max-flow min-cut theorem:

For any graph G = (V, E) with capacities on its edges,

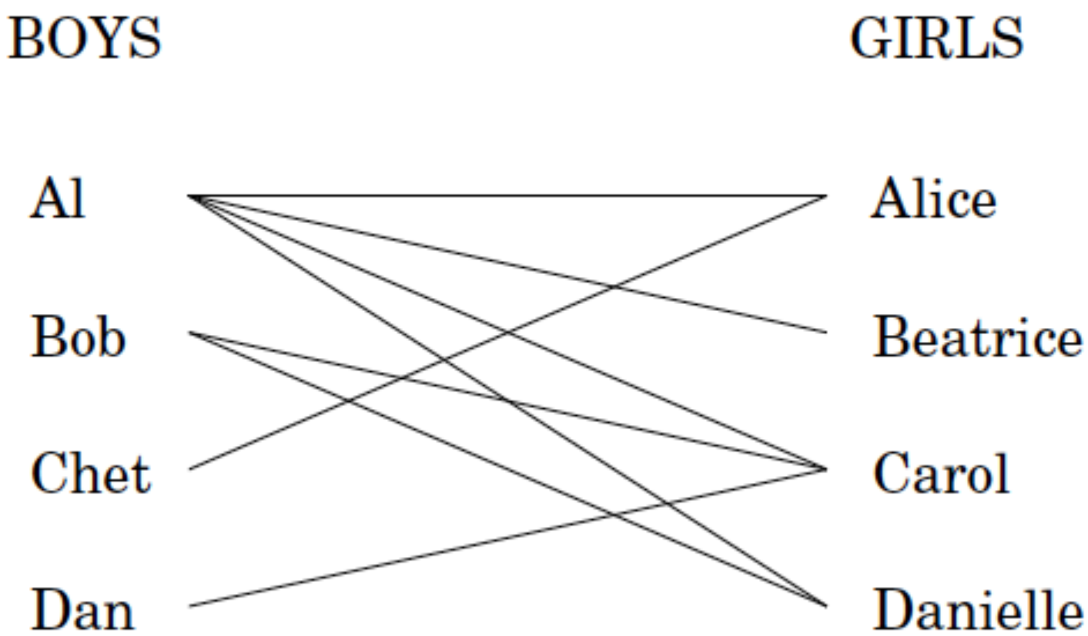
max flow = capacity of minimum s-t cut

In our example, the cut (L, R) shows immediately that the flow of 7 units in Figure (b) is optimal!

The proof of the max-flow min-cut theorem can be done using the LP formulation of the problem (in particular using LP-Duality)

Matching in Bipartite Graphs

Consider a bipartite graph G = (U, V, E), with |U| = |V| = n

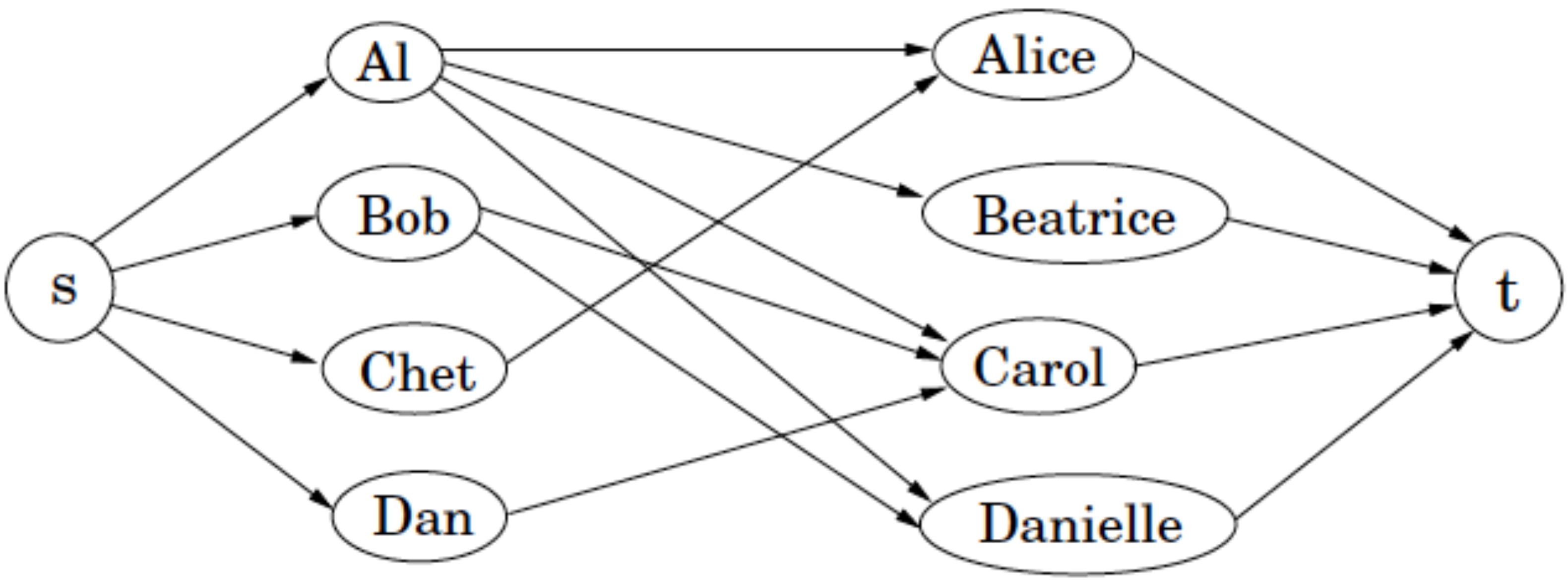


Q1: Find a maximum matching in the graph

Or we may be interested in asking:
Q2: Is there a perfect matching in G?

Matching in Bipartite Graphs

We will reduce this to a max-flow problem, and hence to Linear Programming



- Orient all edges from left to right
- Add a source node s, connect it to all of U
- Add a sink node t, connect all of V to t
- Capacities: set them to 1 for all edges

Matching in Bipartite Graphs

Hence:

- a maximum matching for bipartite graphs can be computed in polynomial time
- The graph has a perfect matching if and only if the max flow in the modified graph equals n

But wait a minute...
What if the max flow assigns a flow of 0.65 to an edge?

Fortunately this can be avoided

Theorem: If all the capacities of a graph are integral, then there is an integral optimal flow and there are algorithms that compute such an integral optimal flow

Vertex Cover and Set Cover

Vertex Cover (VC)

Recall the (optimization) version:

VERTEX COVER (VC):

I: A graph G = (V,E)

Q: Find a cover $C \subseteq V$ of maximum size, i.e., a set $C \subseteq V$, s.t. $\forall$ (u, v) $\in$ E, either u $\in$ C or v $\in$ C (or both)

Weighted version:

WEIGHTED VERTEX COVER (WVC):

I: A graph G = (V,E), and a weight w(u) for every vertex u$\in$V

Q: Find a subset $C \subseteq V$ covering all edges of G, s.t. $W = \sum_{u \in C} w(u)$ is minimized

Many different approximation techniques have been "tested" on vertex cover

Vertex Cover (VC)

We will focus first on the unweighted version

Natural greedy algorithms: start picking nodes according to some criterion until all edges are covered

1st approach:

Greedy-any-node

C := $\emptyset$;

while E $\neq \emptyset$ do

{ choose arbitrarily a vertex u $\in$ V;

delete u and its incident edges from G;

Add u to C }

What is the approximation ratio this algorithm ?

Vertex Cover (VC)

2nd natural approach: start picking nodes and at each step choose the node with the maximum degree

Greedy-best-node

$C := \emptyset$;

while $E \neq \emptyset$ do

{ choose the vertex $u \in V$ with the largest degree; (break ties arbitrarily)

delete u and its incident edges from G;

Add u to C }

Theorem: Greedy-best-node is an O(log n)-approximation algorithm

Vertex Cover (VC)

- The O(logn) ratio of Greedy-best-node is tight
- Can you find an example?

Q: Are there constant factor approximation algorithms?

Vertex Cover (VC)

A different approach:

- Again we will resort to matching
- Let M be any matching in the graph
- Observation: OPT ≥ |M|
  - The optimal solution needs at least one vertex to cover each of the matched edges
- But we cannot just pick any matching, since it may not be a cover

<u>Matching-based VC</u>

C ≠ ∅;

Find a maximal matching M;

For every (u, v) ∈ M, add both u and v to C

Output C

Theorem: Matching-based VC is a 2-approximation algorithm

Vertex Cover (VC)

Theorem: Matching-based VC is a 2-approximation algorithm

Proof:

Claim: The solution returned by the algorithm is a vertex cover

- Suppose not
- Then there is an uncovered edge (u, v)
- But then we could add this edge to the matching M
- Contradiction with the fact that M is a maximal matching

Cost of the solution: $|C| = 2\ |M| \leq 2\ OPT$ (by the observation)

Hence a 2-approximation

Is it easy to find a maximal matching?

Trivial! Keep adding edges until it is not feasible to add more

Vertex Cover (VC)

A way to implement the maximal matching based algorithm

Greedy-any-edge

$C := \emptyset$;

while $E \neq \emptyset$ do

{ choose arbitrarily an edge (u,v) $\in E$;

delete u and v and their incident edges from G;

Add u and v to C; }

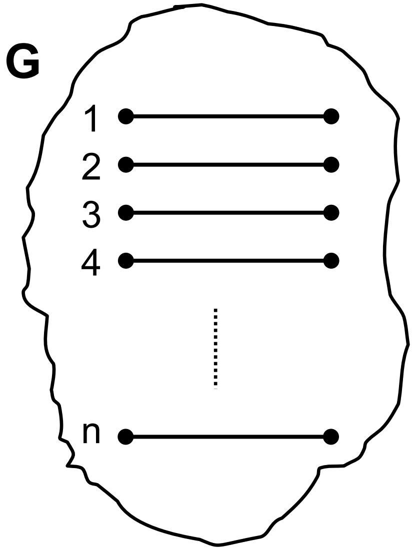
The edges selected by the algorithm form a maximal matching (no 2 edges share a common vertex)

Note: In contrast to greedy-any-node, greedy-any-edge achieves a constant factor approximation

Vertex Cover (VC)

Tightness of the 2-approximation

Example:



C = 2n

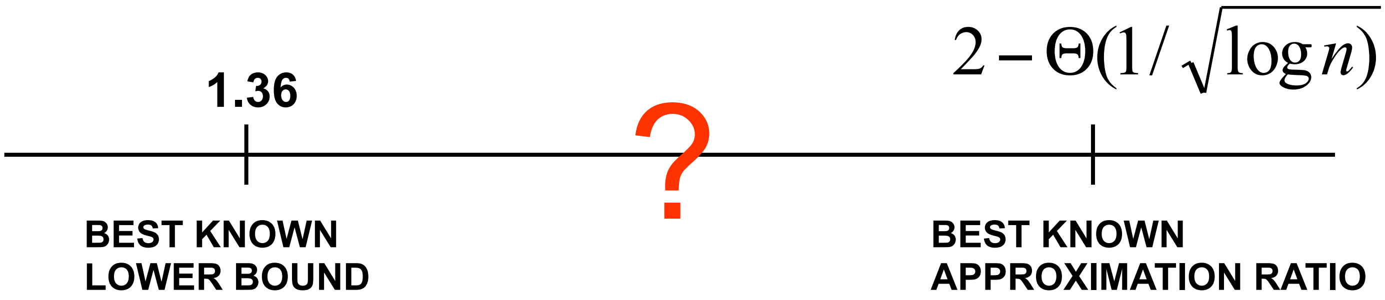
OPT = n

Vertex Cover (VC)

Greedy-any-edge is almost the best known for VC

Is there a better approximation algorithm ?

We know a lower bound of 1.36 on the approximation factor for VC, i.e.,
Unless P=NP, VC cannot be approximated with a ratio smaller than 1.36



Big open problem!!

Weighted Vertex Cover (WVC)

- The algorithms we have seen so far do not apply to the weighted case
- A maximal matching does not guarantee anything about the total weight of the solution returned
- Can we have constant approximations here as well?
- For this, we will resort to techniques from Linear and Integer Programming

Integer Programming Formulations

- Modeling a problem as an **Integer Program** (IP) (also referred to as Integer Linear Program):
- Same as with Linear Programs but (maybe some of) the variables take integer values
- Assign a binary variable x_i to candidate items that can be included in a solution
- Interpretation: $x_i = \begin{cases} 1, & \text{if item } i \text{ is in a solution} \\ 0, & \text{otherwise} \end{cases}$

Examples:

<u>Weighted Vertex Cover</u>

min $\Sigma_u w(u)\, x_u$

s.t.

$x_u + x_v \geq 1 \quad \forall (u, v) \in E$

$x_u \in \{0,1\} \quad \forall u \in V$

<u>0-1 KNAPSACK</u>

max $\Sigma_i v_i x_i$

s.t.

$\Sigma_i w_i x_i \leq W$

$x_i \in \{0,1\} \quad \forall i \in \{1,\ldots,n\}$

Linear Programming Relaxations

- We cannot hope to solve the integer programs
- Integer Programming is NP-hard
- But we can relax the integrality constraints to get an LP

LP relaxations:

Weighted Vertex Cover

min $\Sigma_u w(u) x_u$
s.t.
$x_u + x_v \geq 1 \quad \forall (u, v) \in E$
$x_u \in [0,1] \quad \forall u \in V$

0-1 KNAPSACK

max $\Sigma_i v_i x_i$
s.t.
$\Sigma_i w_i x_i \leq W$
$x_i \in [0,1] \quad \forall i \in \{1,\ldots,n\}$

Main observation:

- For minimization problems: LP-OPT ≤ IP-OPT = OPT
- For maximization problems: LP-OPT ≥ IP-OPT = OPT
  - In the LP, we are optimizing over a larger space of possible solutions

Linear Programming Relaxations

- Solving the LP, we get a fractional solution
- But what can we do with it? It is after all not a valid solution for our original problem
- E.g., what is the meaning of having x_u = 0.8 for a vertex cover instance?
- LP-rounding: the process of constructing an integral solution to the original problem, given an optimal fractional solution of the corresponding LP
- The process is problem-specific, but there are some general guidelines
- A natural first idea: objects with a high fractional value may be preferred (e.g., if in the LP, x_u = 0.8, it may be beneficial to include vertex u in an integral solution)

Linear Programming Relaxations

General scheme for LP rounding:

1. Write down an IP for the problem we want to solve
2. Convert IP to LP
3. Solve LP in O(poly) time to obtain a fractional solution
4. Find a way to convert the fractional solution to an integral one
   - The constructed solution should not lose much in the objective function from LP-OPT
5. Prove that the integral solution has a good approximation guarantee
   - Exploit the main observation to derive bounds with respect to OPT

LP Rounding for WVC

1. First solve:

$$\min \quad \Sigma_u\, w(u)\, x_u$$
$$\text{s.t.}$$
$$x_u + x_v \geq 1 \quad \forall\, (u, v) \in E$$
$$x_u \in [0,1] \quad \forall\, u \in V$$

2. Let $\{x_v\}_{v \in V}$ be the optimal fractional solution

3. Rounding: Include in the cover all vertices v, for which $x_v \geq ½$
Rationale: Vertices with a high fractional value are more likely to be important for the cover. We also stay "close" in value to LP-OPT

Theorem: The LP rounding algorithm achieves a 2-approximation for the Weighted Vertex Cover problem

Rounding for WVC

Let C be the collection of vertices picked

Claim 1: C is a valid vertex cover

- We started with a feasible LP solution
- Hence, for every edge (u, v), $x_u + x_v \geq 1$
- Thus either $x_u \geq ½$ or $x_v \geq ½$
- By the way we constructed our solution, either u or v belongs to C
- Hence, every edge is covered

Rounding for WVC

Claim2: C achieves a 2-approximation for WVC

Let C be the collection of vertices picked
C corresponds to the integral solution: $y_u = 1$ if $u \in C$, $y_u = 0$ otherwise
Note: $y_u \leq 2\, x_u$, for every $u \in V$

Given this and the main observation:

$$SOL = \sum_{u \in C} w(u) = \sum_{u \in V} w(u) \cdot y_u \leq \sum_{u \in V} w(u) \cdot 2 \cdot x_u = 2 \cdot \text{LP-OPT} \leq 2 \cdot OPT$$

Set Cover

SET COVER (SC):

I: a set U of n elements

a family F = {S_1, S_2, …,S_m} of subsets of U

Q: Find a minimum size subset C ⊆ F covering all elements of U, i.e.:

$$\bigcup_{S_i \in C} S_i = U \text{ and } |\mathrm{C}| \text{ is minimized}$$

Weighted version:

WEIGHTED VERTEX COVER (WSC):

I: a set U of n elements

a family F = {S_1, S_2, …, S_m} of subsets of U

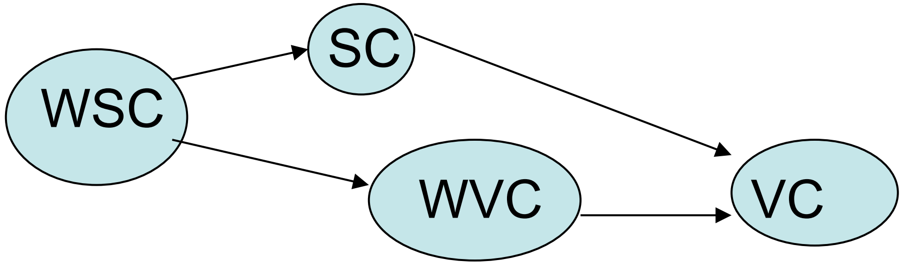
a weight $w(S_i)$ for each set S_i

Q: Find a minimum weight subset C ⊆ F covering all elements of U, i.e.,

$$\bigcup_{S_i \in C} S_i = U \text{ and } \mathrm{W} = \sum_{S_i \in C} w(S_i) \text{ is minimized}$$

Set Cover vs Vertex Cover

- (weighted) vertex cover is a special case of (weighted) set cover
- Consider a vertex cover instance on a graph G = (V, E)
- Let U = E (need to cover the edges)
- One set per vertex, $S_u = \{(u,v) \mid (u,v) \in E\}$, $|F| = |V|$
- In the weighted case, weight of set $S_u = w(u)$



Set Cover vs Vertex Cover

- f_u = frequency of an element $u \in U$ = # of sets S_i that u belongs to
- $f = \max_{u \in U} \{ f_u \}$ = frequency of the most frequent element
- If f=2 (and $w(S_i)$ =1) then (W)SC reduces to (W)VC:
  - $G=(V,E)$, $V= F$, $E= \{ (u,v) \mid S_u \cap S_v \neq 0 \}$

There are approximation algorithms for WSC,
and hence, for SC, WVC and VC,
of ratios:

- O(log n) (n: the size of the universe U) by a greedy approach
- f, using an LP rounding approach
  - Extending the 2-approximation for vertex cover

Weighted Set Cover (WSC)

In a similar spirit as for Vertex Cover:

Greedy-best-set

C := $\emptyset$;

while C $\neq$ U do

{ choose the best set S;

remove S from F;

C := C U S ; }

C: elements covered before iteration i

S: Set chosen at iteration i

Q: What does "best set" mean ?

S covers |S-C| new elements

Covering those elements costs *w(S)*

Every element x $\in$ S essentially costs $\frac{w(S)}{|S-C|} = p(x)$ = "cost-effectiveness" of S

Best set: the set with the smallest cost-effectiveness

Weighted Set Cover (WSC)

Greedy-best-set (cont.)

Let $x_{1,} x_{2,} \ldots, x_{k,} \ldots, x_n$ be the order in which the elements of U are covered

$S_{1,} S_{2,} \ldots\ S_{i,} \ldots$ be the order in which sets are chosen by the algorithm

Suppose set S_i covers element x_k

Claim: $p(x_k) \leq \dfrac{OPT}{n-k+1}$

$C = \bigcup_{j=1}^{i-1} S_j$ elements covered by iterations 1,2,…,i-1

- U-C: uncovered elements before iteration i
- |U-C| ≥ n-k+1, since element x_k is covered in iteration i

Weighted Set Cover (WSC)

- These elements of U-C are covered in the optimal solution by some sets at a cost of at most OPT
- Among them there must be one set with cost-effectiveness at most

$$\leq \frac{OPT}{|U-C|} \leq \frac{OPT}{n-k+1}$$

- the set S_i was picked by the algorithm as the set with the best cost-effectiveness at that moment (and it covered x_k)
- that is $p(x_k) \leq \dfrac{OPT}{n-k+1}$

$$W = \sum_{k=1}^{n} p(x_k) \leq \sum_{k=1}^{n} \frac{OPT}{n-k+1} = OPT \sum_{i=1}^{n} \frac{1}{k} = OPT \cdot H_n = O(\log n) OPT$$

Rounding for WSC

LP relaxation for Set Cover:

$$
\begin{aligned}
&\min \sum_{S} x_S \\
&s.t. \\
&\qquad \sum_{u:u\in S} x_S \geq 1, \quad \forall u \in U \\
&\qquad x_S \geq 0, \qquad\qquad \forall S \in F
\end{aligned}
$$

Q: How should we round a fractional solution?

Rounding for WSC

LP rounding:

- Solve the LP relaxation
- Fractional solution **x** = $\{x_S\}_{s \in F}$ of cost LP-OPT
- Rounding: if $x_S \geq 1/f$, then include S in the cover

Theorem: The LP Rounding algorithm achieves an approximation ratio of f for the WSC problem

Rounding for WSC

Proof:
Let C be the collection of sets picked

Claim 1: C is a valid set cover

Assume not

- Then there exists some u that is not covered
- => For each set S for which $u \in S$, $x_S < 1/f$
- But then:

$$\sum_{S:u\in S} x_S < \frac{1}{f} \mid \{S : u \in S\} \mid = \frac{1}{f} f_u \leq \frac{1}{f} f = 1$$

- a contradiction since we found a violated LP constraint

Rounding for WSC

Proof:
Let C be the collection of sets picked

Claim 2: C achieves an f-approximation

Proof very similar to the proof for WVC