CSC2420: Algorithm Design, Analysis and Theory Fall 2017

Allan Borodin and Nisarg Shah

October 11, 2017

Lecture 5

Announcements:

- The first assignment is due next week, October 18, at 1:00 PM
- The Markus system requires a specific time of day and **does not allow any lateness**. This is strict as the system will not accept a submission even a minute late.
- I have posted two initial questions for Assignment 2. The tenetative due date is November 8.

Todays agenda:

- Continue discussion of local search.
- Max flow, Ford-Fulerson max flow-min cut theorem and algorithm, applications of max flow
- Start IP/LP discussion

Oblivious and non-oblivious local search for $k+1$ claw free graphs

- We again consider the maximum weighted independent set problem in a $k+1$ claw free graph. (Recall the set packing problem the argument generalizing the approximation ratio for the k set packing problem.)
- The standard greedy algorithm and the 1-swap oblivious local search both achieve a $\frac{1}{k}$ approximation for the WMIS in $k+1$ claw free graphs. Here we define an "ℓ-swap" oblivious local search by using the neighbourhood defined by bringing in a set S of up to ℓ vertices and removing all vertices adjacent to S.
- For the unweighted MIS, Halldórsson shows that a a 2-swap oblivious local search will yield a $\frac{2}{k+1}$ approximation.

Berman's [2000] non-oblivious local search

- For $k + 1$ claw free graphs and the weighted MIS, the "ℓ-swap" oblivious local search results (essentially) in an $\frac{1}{k}$ locality gap for any constant ℓ.
- Chandra and Halldóssron [1999] show that by first using a standard greedy algorithm to initialize a solution and then using a "greedy" k-swap oblivious local search, the approximation ratio improves to $\frac{3}{2k}$.
- Can we use non-oblivious local search to improve the locality gap? Once again we ask (as we did for exact Max-2-Sat), given two solutions V_1 and V_2 having the same weight, when is one better than the other?

Berman's [2000] non-oblivious local search

- For $k+1$ claw free graphs and the weighted MIS, the "ℓ-swap" oblivious local search results (essentially) in an $\frac{1}{k}$ locality gap for any constant ℓ.
- Chandra and Halldóssron [1999] show that by first using a standard greedy algorithm to initialize a solution and then using a "greedy" k-swap oblivious local search, the approximation ratio improves to $\frac{3}{2k}$.
- Can we use non-oblivious local search to improve the locality gap? Once again we ask (as we did for exact Max-2-Sat), given two solutions V_1 and V_2 having the same weight, when is one better than the other?
- Intuitively, if one vertex set V_1 is small but vertices in V_1 have large weights that is better than a solution with many small weight vertices.

Berman's [2000] non-oblivious local search

- For $k+1$ claw free graphs and the weighted MIS, the "ℓ-swap" oblivious local search results (essentially) in an $\frac{1}{k}$ locality gap for any constant ℓ.
- Chandra and Halldóssron [1999] show that by first using a standard greedy algorithm to initialize a solution and then using a "greedy" k-swap oblivious local search, the approximation ratio improves to $\frac{3}{2k}$.
- Can we use non-oblivious local search to improve the locality gap? Once again we ask (as we did for exact Max-2-Sat), given two solutions V_1 and V_2 having the same weight, when is one better than the other?
- Intuitively, if one vertex set V_1 is small but vertices in V_1 have large weights that is better than a solution with many small weight vertices.
- Berman chooses the potential function $g(S) = \sum_{v \in S} w(v)^2$. Ignoring some small ϵ's, his k-swap non-oblivious local search achieves a locality gap of $\frac{2}{k+1}$ for WMIS on $k+1$ claw-free graphs.

Some (almost) concluding comments (for now) on local search

- For the metric k-median problem, until recently, local search gave the best approximation algorithm. Using a p-flip (of facilities) neighbourhood, Arya et al (2001) obtain a $3 + 2/p$ approximation which yields a $3 + \epsilon$ approximation running in time $O(n^{2/\epsilon})$.
- Li and Svensson (2013) obtained a $(1 + \sqrt{3} + \epsilon) \approx 2.732 + \epsilon$ LP-based approximation running in time $O(n^{1/\epsilon^2})$. Surprisingly, they show that an α approximate "pseudo solution" using $k + c$ facilities can be converted to an $\alpha + \epsilon$ approximate solution running in $n^{O(c/\epsilon)}$ times the complexity of the pseudo solution. The latest improvement is a $2.633 + \epsilon$ approximation by Ahmadian et al (2017).
- An interesting (but probably difficult) open problem is to use non oblivious local search for the metric k-median, facility location, or k-means problems. These well motivated clustering problems play an important role in operations research, CS algorithm design and machine learning.

End of current concluding remarks on local search

- Perhaps the main thing to mention now is that local search is the basis for many practical algorithms, especially when the idea is extended by allowing some well motivated ways to escape local optima (e.g. simulated annealing, tabu search) and combined with other paradigms.
- Although local search with all its variants is viewed as a great "practical" approach for many problems, local search is not often theoretically analyzed. It is not surprising then that there hasn't been much interest in formalizing the method and establishing limits.
- We will be discussing paradigms relating to Linear Programming (LP). LP is often solved by some variant of the simplex method, which can be thought of as a local search algorithm, moving from one vertex of the LP polytope to an adjacent vertex.
- Our next "paradigm" is max flow and flow based algorithms and max flow is often solved by some variant of the Ford Fulkerson method which also can be thought of as a local search algorithm.

Ford Fulkerson max flow based algorithms

A number of problems can be reduced to the max flow problem. As suggested, max flow itself can be viewed as a local search algorithm.

Flow Networks

A flow network $\mathcal{F} = (G, s, t, c)$ consists of a "bi-directional" graph $G = (V, E)$, a source s and terminal node t, and c is a non-negative real valued (*capacity*) function on the edges.

What is a flow

A flow f is a real valued function on the edges satisfying the following properties:

1. $f(e) \leq c(e)$ for all edges e (capacity constraint)
2. $f(u, v) = -f(v, u)$ (skew symmetry)
3. For all nodes u (except for s and t), the sum of flows into (or out of) u is zero. (Flow conservation).
   Note: this is the "flow in = flow out" constraint for the convention of only having non negative flows.

The max flow problem

- The goal of the max flow problem is to find a valid flow that maximizes the flow out of the source node s. As we will see this is also equivalent to maximizing the flow in to the terminal node t. (This should not be surprising as flow conservation dictates that no flow is being stored in the other nodes.) We let $val(f) = |f|$ denote the flow out of the source s for a given flow f.
- We will study the Ford Fulkerson augmenting path scheme for computing an optimal flow. I am calling it a scheme as there are many ways to instantiate this scheme although I dont view it as a general paradigm in the way I view (say) greedy and DP algorithms.
- I am assuming that many people in the class have seen the Ford Fulkerson algorithm so I will discuss this quickly. I am following the development of the model and algorithm as in Cormen et al (CLRS), second edition. That is, we have negative flows which simplifies the analysis but may be less intuitive.

A flow f and its residual graph

- Given any flow f for a flow network $\mathcal{F} = (G, s, t, c)$, we can define the residual graph $G_f = (V, E(f))$ where $E(f)$ is the set if all edges e having *positive* residual capacity ; i.e. the residual capacity of e wrt to f is $c_f(e) = c(e) - f(e) > 0$.
- Note that $c(e) - f(e) \geq 0$ for all edges by the capacity constraint. Also note that with our convention of negative flows, even a zero capacity edge (in G) can have residual capacity.
- The basic concept underlying Ford Fulkerson is that of an augmenting path which is an $s - t$ path in G_f. Such a path can be used to augment the current flow f to derive a better flow f'.
- Given an augmenting path π in G_f, we define its residual capacity wrt f as $c_f(\pi) = \min\{c_f(e) | e \text{ in the path } \pi\}$.

The Ford Fulkerson scheme

Ford Fulkerson

```
$f := 0$ ;
$G_f := G$   %initialize
**While** there is an augmenting path in $G_f$
    Choose an augmenting path $\pi$
    $\tilde{f} := f + f_\pi; f := \tilde{f}$   % Note this also changes $G_f$
**End While**
```

I call this a scheme rather than a well specified algorithm since we have not said how one chooses an augmenting path (as there can be many such paths)

The max flow-min cut theorem

Ford Fulkerson Max Flow-Min Cut Theorem

The following are equivalent:

1. f is a max flow
2. There are no augmenting paths wrt flow f; that is, no $s - t$ path in G_f
3. $val(f) = c(S, T)$ for some cut (S, T) ; hence this cut (S, T) must be a min (capacity) cut since $val(f) \leq c(S, T)$ for all cuts.

Hence the name max flow $(=)$ min cut

Comments on max flow - min cut theorem

- As previously mentioned, the Ford Fulkerson algorithms can be viewed as local search algorithms.
- This is a rather unusual local search algorithm in that any local optimum is a global optimum.
- Suppose we have a flow network in which all capacities are integral. Then :
  1. Any Ford Fulkerson implementation must terminate.
  2. If the sum of the capacities for edges leaving the source s is C, then the algorithm terminates in at most C iterations and hence with complexity at most $O(mC)$.
  3. Ford Fulkerson implies that there is an optimal integral flow. (There can be other non integral optimal flows.)

Good and bad ways to implement Ford Fulkerson

- There are bad ways to implement the networks such that
  1. For some algebraic capacities, the algorithm may not terminate.
  2. There are networks with integer capacities where the algorithm uses exponential (in representation of the capacities) time to terminate.
- There are various ways to implement Ford-Fulkerson so as to achieve polynomial time. Edmonds and Karp provided the first polynomial time algorithm showing that shortest length augmenting paths yields time bound $O(|V| \cdot |E|^2)$. Dinitz obtains time complexity $O(|V|^2|E|)$ and also has the advantage of leading to a $O(m\sqrt{n})$ time bound for unweighted bipartite matching. I think the best known worst case time for max flow is the preflow-push-relabel algorithm of Goldberg and Tarjan with time $O(|V| \cdot |E| \; polylog(|E|)$ or maybe $O(|V| \cdot |E|)$.
- Although not strongly polynomial time. Madry[2016] has made significant progress in developing a max flow algorithm with time bound $O(m^{10/7}C^{1/7})$ where C is the maximum edge capacity. For "sparse graphs" and "small" capacities, this is the best known time bound.

The Dinitz (sometimes written Dinic) algorithm

- Given a flow f, define the leveled graph $L_f = (V', E')$ where $V' = \{v | v \text{ reachable from } s \text{ in } G_f\}$ and $(u, v) \in E'$ iff $level(v) = level(u) + 1$. Here $level(u) =$ length of shortest path from s to u.
- A blocking flow $\tilde{f}$ is a flow such that every s to t path in L_f has a saturated edge.

The Dinitz Algorithm

```
Initialize $f(e) = 0$ for all edges $e$
While $t$ is reachable from $s$ in $G_f$ (else no augmenting path)
    Construct $L_f$ corresponding to $G_f$
    Find a blocking flow $\hat{f}$ wrt $L_f$ and set $f := f + \hat{f}$
End While
```

The run time of Dinitz' algorithm

Let $m = |E|$ and $n = |V|$

- The algorithm halts in at most $n - 1$ iterations (i.e. blocking steps).
- The residual graph and the levelled graph can be computed in time $O(m)$ with breadth first search and using depth first search we can compute a blocking path in time $O(mn)$. Hence the total time for the Dinitz blocking flow algorithm is $O(mn^2)$
- A unit network is one in which all capacities are in $\{0,1\}$ a nd for each node $v \neq s, t$, either v has at most one incoming edge (i.e. of capacity 1) or at most one outgoing edge. In a unit network, the Dinitz algorithm terminates within $2\sqrt{n}$ iterations and hence on such a network, a max flow can be computed in time $O(m\sqrt{n})$ (Hopcroft and Karp [1973]. As we next show, this immediately leads to a $O(m\sqrt{n})$ time algorithm for maximum unweighted bipartite matching. **Note:** Better bipartite bounds are known for "sparse" graphs (Madry's $O(m^{10/7})$ and "dense graphs" (Mucha and Sankowsky'a n^{ω}) randomized algorithm where ω is the matrix multiplication algorithm.

Application to unweighted bipartite matching

- We can transform the maximum bipartite matching problem to a max flow problem.
  Namely, given a bipartite graph $G = (V, E)$, with $V = X \cup Y$, we create the flow network $\mathcal{F}_G = (G', s, t, c)$ where
  - $G' = (V', E')$ with $V' = V \cup \{s, t\}$ for nodes $s, t \notin V$
  - $E' = E \cup \{(s, x) | x \in X\} \cup \{(y, t) | y \in Y\}$
  - $c(e) = 1$ for all $e \in E'$.

.

Claim: Every matching M in G gives rise to an integral flow f_M in $\mathcal{F}_G$ with $val(f_M) = |M|$; conversely every integral flow f in $\mathcal{F}_G$ gives rise to a matching M_f in G with $|M| = val(f)$.

- This implies bipartite matching can be computed in time $O(m\sqrt{n})$ using the Hopcroft and Karp adaption of the blocking path algorithm.
- Similar ideas allow us to compute the maximum number of edge (or node) disjoint paths in directed and undirected graphs.

Additional comments on maximum bipartite matching

- There is a nice terminology for augmenting paths in the context of matching. Let M be a matching in a graph $G = (V, E)$. A vertex v is *matched* if it is the end point of some edge in M and otherwise if is *free*. A path π is an alternating path if the edges in π alternate between M and $E - M$.
- Abusing terminology briefly, an augmenting path (relative to a matching M) is an alternating path that starts and ends in a free vertex. An augmenting path in a graph shows that the matching is not a maximum and can be immediately improved.
- Clearly the existence of an augmenting path in a bipartite graph G corresponds to an augmenting path in the flow graph $\mathcal{F}_G$ used to show that bipartite matching reduce to flows.

The weighted bipartite matching problem

- Can the flow algorithm for unweighted bipartite matching be modified for weighted bipartite matching?
- The obvious modification would set the capacity of $< x, y > \in E$ to be its weight $w(x, y)$ and the capacity of any edge $< s, x >$ could be set to $\max_y\{w(x, y)\}$ and similarly for the weight of edges $< y, t >$.
- Why doesnt this work?
- It is true that if G has a matching of total weight W then the resulting flow network has a flow of value W.
- But the converse fails! Why?
- We will hopefully return to the weighted bipartite matching problem (also known as the *assignment problem*) discussing the optimal "Hungarian method formalized by Kuhn [1955] and attributed by Kuhn to Konig and Egevary [1931].
- This method is intimately tied to linear programming duality.
- We will also discuss the online matching problem and variants.

The $\{0,1\}$ metric labelling problem.

We consider one more application of max flow-min cut, the $\{0,1\}$ metric labelliing problem, discussed in sections 12.6 and 7.10 of the KT text.

- This problem is a special case of the following more general metric labelling problem defined as follows:
- The input is an edge weighted graph $G = (V, E)$, a set of labels $L = \{a_1, \ldots, a_r\}$ in a metric space with distance metric d, and functions $w : E \rightarrow \mathbb{R}^{\geq 0}$ and $p : V \times L \rightarrow \mathbb{R}^{\geq 0}$.
- $\beta(u, a_j)$ is the benefit of giving label a_j to node u.
- Goal: Find a labelling $\lambda : V \rightarrow L$ of the nodes so as to maximize

$$\sum_u \beta(u, \lambda(u)) - \sum_{(u,v)\in E} w(u, v) \cdot d((\lambda(u), \lambda(v))$$

The $\{0,1\}$ metric labelling problem.

We consider one more application of max flow-min cut, the $\{0,1\}$ metric labelliing problem, discussed in sections 12.6 and 7.10 of the KT text.

- This problem is a special case of the following more general metric labelling problem defined as follows:
- The input is an edge weighted graph $G = (V, E)$, a set of labels $L = \{a_1, \ldots, a_r\}$ in a metric space with distance metric d, and functions $w : E \rightarrow \mathbb{R}^{\geq 0}$ and $p : V \times L \rightarrow \mathbb{R}^{\geq 0}$.
- $\beta(u, a_j)$ is the benefit of giving label a_j to node u.
- Goal: Find a labelling $\lambda : V \rightarrow L$ of the nodes so as to maximize

$$\sum_u \beta(u, \lambda(u)) - \sum_{(u,v)\in E} w(u, v) \cdot d((\lambda(u), \lambda(v))$$

- For example, the nodes might represent documents, the labels are topics, and the edges are links between documents weighted by the importance of the link.
- When there are 3 or more labels, the problem is NP-hard even for the case of the $\{0,1\}$ metric d where $d(a_i, a_j) = 1$ for $a_i \neq a_j$.

The labelling problem with 2 labels

- When there are only 2 labels, the only metric is the $\{0,1\}$ metric.
- While the labelling problem is NP-hard for 3 or more labels (even for the $\{0,1\}$ metric), it is solvable in polynomial time for 2 labels by reducing the problem to the min cut problem. This is what is being done in Section 7.10 of the KT text for a special graph relating to pixels in an image.
- Later we may see an approximation algorithm for the $\{0,1\}$ metric with 3 or more labels that uses local search and a reduction to min cut.
- Informally, the idea is that we can construct a flow network such that the nodes on the side of the source node s will correspond to (say) nodes labled a and the nodes on the side of the terminal node t will correspond to the nodes labeled b.
- We will place capacities between the source s and other nodes to reflect the cost of a "mislabel" and similarly for the terminal t.
- The min cut will then correspond to a min cost labelling.

The reduction for the two label case

For the two label case (labels {a,b}), we can let $a_u = \beta(u,a)$ and $b_u = \beta(u,b)$.
The goal is to maximize $\sum_{u\in A} a_u + \sum_{v\in B} b_v - \sum_{(u,v)\in A\times B} w(u,v)$
Leting $Q = \sum_{u\in V} a_u + b_u$, the goal is then equivalent to maximizing $Q - \sum_{u\in A} b_u - \sum_{v\in B} a_v - \sum_{(u,v)\in A\times B} w(u,v)$
Equivalently, to minimizing
$\sum_{u\in A} b_u + \sum_{v\in B} a_v + \sum_{(u,v)\in A\times B} w(u,v)$

- We transform this problem to a min cut problem as follows: construct the flow network $\mathcal{F} = (G', s, t, c)$ such that $G' = (V', E')$
- $V' = V \cup \{s,t\}$
- $E' = \{(u,v) | u \neq v \in V\} \cup \{(s,u) | u \in V\} \cup \{(u,t) | u \in V\}$
- $c(i,j) = c(j,i) = p_{i,j}; c(s,i) = a_i; c(i,t) = b_i$

Claim:

For any partition $V = A \cup B$, the capacity of the cut $c(A,B) = \sum_{u\in A} b_i + \sum_{v\in B} a_j + \sum_{(u,v)\in A\times B} w_{u,v}$.

Flow networks with costs

We now augment the definition of a flow network $\mathcal{F} = (G, s, t, c, \kappa)$ where $\kappa(e)$ is the non negative cost of edge e. Given a flow f, the cost of a path or cycle π is $\sum_{e \in \pi} \kappa(e) f(e)$.

MIn cost flow problem

Given a network $\mathcal{F}$ with costs, and given flow f in $\mathcal{F}$, the goal is to find a flow f of minimum cost. Often we are only interested in a max flow min cost.

- Given a flow f, we can extend the definition of an augmenting path in $\mathcal{F}$ to an augmenting cycle which is just a simple cycle (not necessarily including the source) in the residual graph G_f.
- If there is a negative cost augmenting cycle, then the flow can be increased on each edge of this cycle which will not change the flow (by flow conservation) but will reduce the cost of the flow.
- A negative cost cycle in a directed graph can be detected by the Bellman Ford DP for the single source shortest path problem.

An application of a min cost max flow

- We return to the weighted interval scheduling problem (WISP) .
- We can optimally solve the m machine WISP using DP but the time for this algorithm is $O(n^m)$.
- Using the local ratio (i.e. priority stack) algorithm we can approximate the optimal within a factor of $2 - \frac{1}{m}$.
- Arkin and Silverberg [1987] show that the m machine WISP can be reduced efficiently to a min cost problem resulting in an algorithm having time complexity $O(n^2 \log n)$.
- The Arkin and Silverberg reduction is (for me) a subtle reduction which I will sketch.

The Arkin-Silverberg reduction of WISP to a min cost flow problem

- The reduction relies on the role of maximal cliques in the interval graph.
- An alternative characterization of an interval graph is that each job (i.e. interval) is contained in consecutive maximal cliques.
- The goal will be to create a flow graph so that a min cost flow (with value = maximum size clique - m) will correspond to a minimum weight set of intervals whose removal will leave a feasible set of intervals (i.e. can be scheduled on the m machines).
- If $q_1, \ldots, q_r$ are the maximal cliques then the (directed) flow graph will have nodes $v_0, \ldots, v_r$ and three types of edges:
    1. (q_i, q_{i-1}) representing the clique q_i with cost 0 and infinite capacity
    2. If an interval J_i occurs in cliques $q_j, \ldots, q_{j+\ell}$, there is an edge $(v_{j-1}, v_{j+\ell})$ with cost w_i and capacity 1.
    3. For each clique q_i not having maximum size, we have an edge (v_{i-1}, v_i) of cost 0 and capacity maximum clique size - size of q_i. (These can be thought of as "dummy intervals".)

Example of interval graph transformation

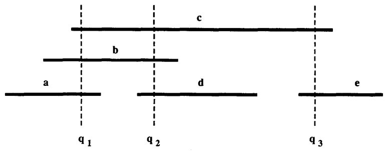

Fig. 1a. The jobs.

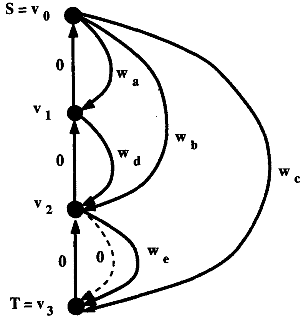

Fig. 1b. The directed graph.

Note: The dotted arc represents a dummy job.

Integer Programming (IP) and Linear Programming (LP)

- We now introduce what is both theoretically and in practice one of the most general frameworks for solving search and optimization problems. Namely, we consider how many problems can be formulated as integer programs (IP). (Later, we will also consider other mathematical programming formulations.)
- Solving an IP is in general an NP hard problem although there are various IP problems that can be solved optimally. Moreover, in practice, many large instances of IP do get solved.
- Our initial emphasis will be on linear program (LP) relaxations of IPs. LPs can be solved optimally in polynomial time as first shown by Khachiyan's ellipsoid method [1979] and then Karmarkar's' [1984] more practical interior point method. In some (many?) cases, Danzig's [1947] simplex method will outperform (in terms of time) the worst case polynomial time methods.
- Smoothed analysis gives an explanation for the success of simplex.
- Open: a strongly polynomial time algorithm for solving LPs?

Some IP and LP concepts

Integer Programs

An IP has the following form:

- Maximize (minimize) $\sum_j c_j x_j$
- subject to $(\sum_j a_{ij} x_j) R_i b_i$ for $i = 1, \ldots, m$
  and where R_i can be $=, \geq, \leq$
- x_j is an integer (or in some prescribed set of integers) for all j

Here we often assume that all parameters $\{a_{ij}, c_j, b_i\}$ are integers or rationals but in general they can be real valued.

An LP has the same form except now the last condition is realized by letting the x_j be real valued. It can be shown that if an LP has only rational parameters then we can assume that the $\{x_j\}$ will be rational.

Canonical LP forms

Without loss of generality, LPs can be formulated as follows:

Standard Form for an LP

• Maximize $\mathbf{c} \cdot \mathbf{x}$	Minimize $\mathbf{c} \cdot \mathbf{x}$
• subject to $A \cdot \mathbf{x} \leq \mathbf{b}$	$A \cdot \mathbf{x} \geq \mathbf{b}$
• $\mathbf{x} \geq 0$	$\mathbf{x} \geq 0$

Slack form

- maximize/minimize $\mathbf{c} \cdot \mathbf{x}$
- subject to $A \cdot \mathbf{x} + \mathbf{s} = \mathbf{b}$
- $\mathbf{x} \geq 0; \mathbf{s} \geq 0$

The $\{s_j\}$ variables are called slack variables.

LP relaxation and rounding

- One standard way to use IP/LP formulations is to start with an IP representation of the problem and then relax the integer constraints on the x_j variables to be real (but again rational suffice) variables.
- We start with the well known simple example for the weighted vertex cover problem. Let the input be a graph $G = (V, E)$ with a weight function $w : V \rightarrow \Re^{\geq 0}$. To simplify notation let the vertices be $\{1, 2, \ldots .n\}$. Then we want to solve the following "natural IP representation" of the problem:
  - Minimize $\mathbf{w} \cdot \mathbf{x}$
  - subject to $x_i + x_j \geq 1$ for every edge $(i, j) \in E$
  - $x_j \in \{0, 1\}$ for all j.
- The *intended meaning* is that $x_j = 1$ iff vertex j is in the chosen cover. The constraint forces every edge to be covered by at least one vertex.
- Note that we could have equivalently said that the x_j just have to be non negative integers since it is clear that any optimal solution would not set any variable to have a value greater than 1.

LP rounding for the natural weighted vertex cover IP

- The "natural LP relaxation" then is to replace $x_j \in \{0,1\}$ by $x_j \in [0,1]$ or more simply $x_j \geq 0$ for all j.
- It is clear that by allowing the variables to be arbitrary reals in [0,1], we are admitting more solutions than an IP optimal with variables in $\{0,1\}$. Hence the LP optimal has to be at least as good as any IP solution and usually it is better.
- The goal then is to convert an optimal LP solution into an IP solution in such a way that the IP solution is not much worse than the LP optimal (and hence not much worse than an IP optimum)
- Consider an LP optimum $\mathbf{x}^*$ and create an integral solution $\bar{\mathbf{x}}$ as follows: $\bar{x}_j = 1$ iff $x_j^* \geq 1/2$ and 0 otherwise. We need to show two things:
  1. $\bar{\mathbf{x}}$ is a valid solution to the IP (i.e. a valid vertex cover).
  2. $\sum_j w_j \bar{x}_j \leq 2 \cdot \sum_j w_j x_j^* \leq 2 \cdot \textit{IP-OPT}$; that is, the LP relaxation results in a 2-approximation.

The integrality gap

- Analogous to the locality gap (that we encountered in local search), for LP relaxations of an IP we can define the integrality gap (for a minimization problem) as $\max_{\mathcal{I}} \frac{IP-OPT}{LP-OPT}$; that is, we take the worst case ratio over all input instances $\mathcal{I}$ of the IP optimum to the LP optimum. (For maximization problems we take the inverse ratio.)
- Note that the integrality gap refers to a particular IP/LP relaxation of the problem just as the locality gap refers to a particular neighbourhood.
- The same concept of the integrality gap can be applied to other relaxations such as in semi definite programming (SDP).
- It should be clear that the simple IP/LP rounding we just used for the vertex cover problem shows that the integrality gap for the previously given IP/LP formulation is at most 2.
- By considering the complete graph K_n on n nodes, it is also easy to see that this integrality gap is at least $\frac{n-1}{n/2} = 2 - \frac{1}{n}$.

Integrality gaps and approximation ratios

- When one proves a positive (i.e upper) bound (say c) on the integrality gap for a particular IP/LP then usually this is a constructive result in that some proposed rounding establishes that the resulting integral solution is within a factor c of the LP optimum and hence this is a c-approximation algorithm.
- When one proves a negative bound (say c') on the integrality gap then this is only a result about the given IP/LP. In practice we tend to see an integrality gap as strong evidence that this particular formulation will not result in a better than c' approximation. Indeed I know of no natural example where we have a lower bound on an integrality gap and yet nevertheless the IP/LP formulation leads "directly" into a better approximation ratio.
- In theory some conditions are needed to have a provable statement. For the VC example, the rounding was "oblivious" (to the input graph). In contrast to the K_n input, the LP-OPT and IP-OPT coincide for an even length cycle. Hence this integrality gap is a tight bound on the formulation using an oblivious rounding.