CSC2420: Algorithm Design, Analysis and Theory Spring (or Winter for pessimists) 2017

Allan Borodin

January 30, 2017

Lecture 4

Announcements:

- I have posted all 7 questions for assignment 1. It is due February 13, at the start of clsss.
  When necessary, I will elaborate (and sometimes give hints) for questions during class. So please do ask for any clarifications that are needed!
- I am reserving Thursdays 3-4 for an office hour but I welcome discussing the course whnever I am free. However, I am away this week but on email.
- Can we arrange a 1 or 2 hour lecture outside of the usual time. I am here reading week but not the following week.

Todays agenda:

- Continue Ford Fulkerson and applications of max flow
- Begin Linear Programming (LP)
  - IP/LP rounding
  - LP Duality; primal dual algorithms and analysis

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 termnal 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. This is also clearly equivalent to maximizing the flow in to the terminal node t since 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 consider 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). 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 π
    $\tilde{f} := f + f_{pi}; f := \tilde{f}$ % 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 "max flow = min cut"

Comments on max flow - min cut theorem

- As previously mentioned, Ford Fulkerson algorithms can be viewed as local search algorithms. Also, the neighbouhood is in general of exponential size but it can be efficiently search to find a local improvement or determine that none exists. .
- 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.)
- It follows that if all capacities are rational, then there is an optimal rational valued flow and the algorithm must terminate. Why?

Good and bad ways to implement Ford Fulkerson

- There are bad ways to implement the networks such that
  1. There are networks with non rational capacities where the algorithm does 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 by showing that a shortest length augmenting path yields the time bound $O(|V| \cdot |E|^2)$. For me, the conceptually simplest polynomial time analysis is the Dinitz algorithm which has time complexity $O(|V|^2|E|)$ and also has the advantage of leading to the best known 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 $O(|V| \cdot |E|)$.

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 capaities are in $\{0,1\}$ and 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].

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 $|M| = val(f)$.

- Hence a maximum size bipartite matching can be computed in time $O(m\sqrt{n})$ using the Hopcroft and Karp adatpion 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 return to the weighted biparitite matching problem (also known as the *assignment problem*) discussing both the optimal "Hungarian method formalize by Kuhn [1955] and attributed by Kuhn to Konig and Egevary [1931].
- We will see that this method is intimately tied to linear programming duality.
- We will also discuss the online matching problem and variants.

The metric labelling problem

We consider a problem well motivated by applications in, for example, information retrieval. (See Kleinberg and Tardos text)

The metric labelling problem

Given: graph $G = (V, E)$, a set of labels $L = \{a_1, \ldots, a_r\}$ in a metric space M with distance δ, and a cost function $\kappa : V \times L \to \Re^{\geq 0}$. The goal is to construct an assignment α of labels to the nodes V so as to minimize $\sum_{i \in V} \kappa(i, \alpha(i)) + \sum_{(i,j) \in E} p_{i,j} \cdot \delta(\alpha(i), \alpha(j))$

The idea is that κ represents a cost for labelling the node (e.g. a penalty for a bad classification of a web page), p represents the importance of that edge (e.g. where in a web page a particular link occurs) and δ represents the (basic or unweighted) cost of giving different labels to nodes that are related (e.g. the penalty for different labellings of web pages that are linking to each other or otherwise seem to be discussing similar topics).

The binary label case

- A case of special interest and the easiest to deal with is when the metric is the binary $\{0, 1\}$ metric; that is, $\delta(a, b) = 1$ if $a \neq b$ and 0 otherwise. (When there are only two labels, the binary $\{0, 1\}$ metric is the only metric.)
- The case of two labels suggests that the problem might be formulated as a min cut problem. Indeed this can be done to achieve an optimal algorithm when there are only two labels.
- For more than two labels, the binary metric case becomes NP hard but a there is a 2-approximation via a local search algorithm that uses min cuts to search a local neighbourhood.

The case of two labels

- The problem for two labels can be restated as follows: find a partition $V = A \cup B$ of the nodes so as to minimize $\sum_{i \in A} b_i + \sum_{j \in B} a_j + \sum_{(i,j) \in A \times B} p_{i,j}$
- 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 $c(A, B) = \sum_{i \in A} b_i + \sum_{j \in B} a_j + \sum_{(i,j) \in A \times B} p_{i,j}$.

-

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 of 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 the crreate 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

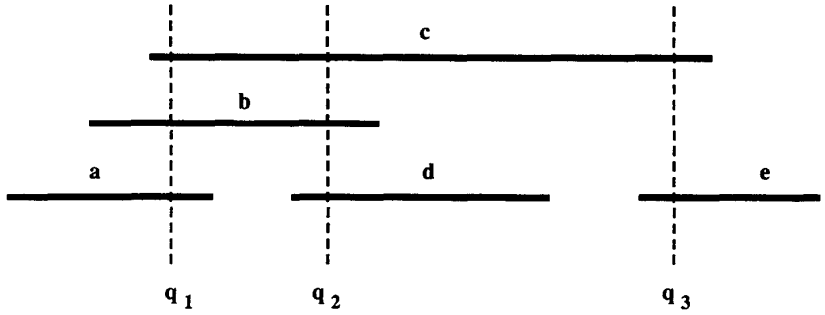

Fig. 1a. The jobs.

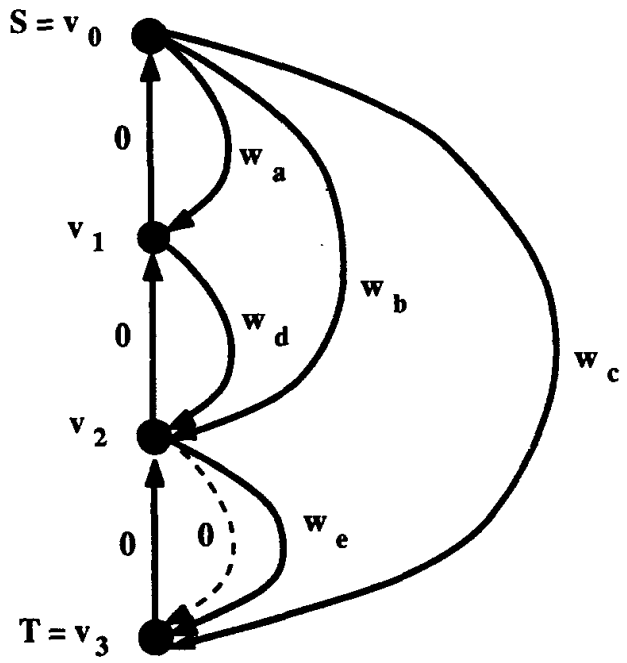

Fig. 1b. The directed graph.

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. $\sum_j w_j \bar{x}_j \leq 2 \cdot \sum_j w_j x_j^* \leq 2 \cdot \mathit{IP\text{-}OPT}$; 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.

Makespan for the unrelated and restricted machine models: a more sophisticated rounding

In the VC example I use the terms "(input) independent rounding" and "oblivious" rounding.)

- We now return to the makespan problem with respect to the unrelated machines model and the special case of the restricted machine model.
- Recall the unrelated machines model where a job j is represented by a tuple $(p_{j,1}, \ldots, p_{j,m})$ where $p_{j,i}$ is the time that job j uses if scheduled on machine i.
- An important scheduling result is the Lenstra, Shmoys, Tardos (LST) [1990] IP/LP 2-approximation algorithm for the makespan problem in the unrelated machine model (when m is part of the input). They also obtain a PTAS for fixed m.

The natural IP and the LP relaxation

The IP/LP for unrelated machines makespan

- Minimize T
- Subject to
  1. $\sum_i x_{j,i} = 1$ for every job j % schedule every job
  2. $\sum_j x_{j,i} p_{j,i} \leq T$ for every machine i % do not exceed makespan
  3. $x_{j,i} \in \{0, 1\}$ % $x_{j,i} = 1$ iff job j scheduled on machine i

- The immdiate LP relaxation is to just have $x_{j,i} \geq 0$
- Even for identical machines (where $p_{j,i} = p_j$ for all i), the integrality gap IG is unbounded since the input could be just one large job with say size T leading to an LP-OPT of T/m and IP-OPT $=$ OPT $= T$ so that the IG $= m$.

Adapting the natural IP

- As in the PTAS for the identical machine makespan PTAS, we use binary search to find an appropriate approximation T for the optimal makespan.
- Given a candidate T, we remove all x_{ji} such that $p_{j,i} > T$ and obtain a "search problem" (i.e. constant or no objective function) for finding $x_{j,i}$ satisfying the IP constraints.
- Once we have found the optimal T for the search problem, the LST algorithm then shows how to use a non-independent rounding to obtain an integral solution yielding a 2-approximation.
- Note: We use the term "rounding" in a very general sense to mean any efficient way to convert the LP solution into an intergral solution.

Sketch of LST rounding for makespan problem

- Using slack form, LP theory can be used to show that if $\mathcal{L}$ is a feasible bounded LP with $m + n$ constraints (not counting the non-negativity constraints for the variables) then $\mathcal{L}$ has an optimal basic solution such that at most $n + m$ of the variables are non-zero.
- It follows how? that there are at most m of the n jobs that have fractional solutions (i.e. are not assigned to a single machine).
- Jobs assigned to a single machine do not need to be rounded; i.e. if $x_{j,i} = 1$ then schedule job j on machine i.
- Construct a bipartite graph between the $y \leq m$ fractionally assigned jobs and the m machines.

The rounding continued

- The goal is then to construct a matching of size y; that, is, the matching dictates how to schedule these fractionally assigned jobs. So it "only" remains to show that this bipartite graph has a matching of size y. Note, of course, this is what makes the "rounding" non-independent .
- The existence of this matching requires more LP theory whereby it can be shown (LST credit Dantzig [1963]) that the connected components of the bipartite graph are either trees or trees with one added edge (and therefore causing a unique cycle).
- The resulting schedule then has makespan at most $2T$ since each fractional job has $p_{j,i} \leq T$ and the LP has guaranteed a makespan at most T before assigning the fractional jobs.