COMP5712 - Combinatorial Optimization

Taught by Sunil Arya

Notes by Aaron Wang

Contents

1 March 26th, 2019

1.1 Flow Decomposition

Lemma 1.1
Let $(G = (V, E), s, t, c)$ be a flow network. Let f be a flow in this network. Then there is a collection of feasible flows $f_1, f_2, \dots f_k$ and a collection of $s - t$ paths $p_1, \dots, p_k$.

- Value of f is equal to the sum of flows f_i.
- flow f_i sends positive flow along edge p_i.
- $k \leq |E|$. (look at lecture notes) for every path, at least one edge will drop off (since once edge gets zero'd, as such we can only have at most $|E|$ paths.

This means the average flow **for the first step** must have $\frac{|f|}{k}$, and the maximum must have more than this amount. This is greater than $\frac{OPT}{|E|}$.

Corollary 1.2
At the beginning, there is a augmenting path from s to t in which each edge has **capacity** $\frac{OPT}{|E|}$, where OPT is the value of the maximum flow.

This is because there must be a $s-t$ path on the residual network for which all edges on that path has capacity greater than equal to $\frac{OPT}{|E|}$ (as we can push this much flow).

Since OPT is changing, even though we need at most $|E|$ iterations, running time will be $|E| \log OPT$.

Theorem 1.3
Assuming a flow network with integer capacities, the fattest-path implementation of Ford-Fulkerson method runs in time at most $(|E| \log (OPT)\, |E| \log |V|)$
$= O(|E|^2 \log |V| \log(OPT)$ which is polynomial.

The time for one iteration to find the largest path is

- $|E|$ to construct the residual network
- $|E| \log |V|$ to run Dijkstra's algorithm to find the path

Proof. Let $m = |E|$ and f_i denote the flow value after i iterations. Let res_i denote the value of optimal flow in the residual network after i iterations.

$$res_i = OPT - f_i.$$

By 1.2, in the $(i+1)$-th iteration, we can find a flow of value greater than or equal to $\frac{res_i}{2|E|}$, as the residual network is a flow network with at most $2|E|$ edges (a forward edge and backward edge for each edge in G). Note that

$$res_{i+1} \leq res_i - \frac{res_i}{2|E|} = res_i \left(1 - \frac{1}{2|E|}\right).$$

As the fattest path has at least $\frac{res_i}{2|E|}$. Now we need to see how many iterations will it take for res_i to be less than or equal to 1 (since they are integer). Note that the factor it drops by is constant. As such we have

$$res_0 = OPT.$$

$$res_t \leq res_{t-1}\left(1 - \frac{1}{2|E|}\right) = \ldots = res_0\left(1 - \frac{1}{2|E|}\right)^t = OPT\left(1 - \frac{1}{2|E|}\right)^t.$$

After $2|E|\ln(OPT)$ iterations

$$res_t \approx OPT\left(1 - \frac{1}{2|E|}\right)^{2|E|\ln OPT} \approx \left(\frac{1}{e}\right)^{\ln OPT} = 1.$$

□

As such for $t \approx 2|E|\ln OPT + 1$, $res_t < 1 \implies res_t = 0$. Look at formal proof in notes.

The $\log(OPT)$ factor in the running time is awkward for many people, so we will try to remove it.

1.2 Edmonds-Karp Algorithm

Definition 1.4 (Strongly Polynomial vs Weakly Polynomial)**.** An algorithm runs in **strongly** poly time if, assuming unit time arithmetic operations (e.g. $+, -$), the running time is polynomial in the # of numerical operations given as input (polynomial only in $|V|, |E|$).

Note that our algorithm is not strongly polynomial - we say that it is weakly polynomial. (polynomial in the number of bits in the problem size).

This distinction is only applicable to problems dealing with integers. One problem where this is relevant is in linear programming. As of now, there are no strongly polynomial time algorithm known yet. All known poly-time algorithms are weakly polynomial (ellipsoid method, integer point methods are weakly polynomial). However, for NP-Complete perspective, both are polynomial in the input size.

Edmonds-Karp Algorithm is a:

- Strongly polynomial
- Specific implementation of FF method
- At each iteration, choose the path in the residual network with the smallest # of edges. Each iteration will take $|E|$ time (by BFS).

Theorem 1.5

If, at a certain iteration, the length of a shortest $s - t$ path is ℓ then at every subsequent iteration, it is $\geq \ell$. Furthermore, after at most $|E|$ iterations, then the length of the shortest $s - t$ path becomes $\geq \ell + 1$.

Note that ℓ can only stay the same or increase. In addition, at each ℓ you can only stay at the same length for $|E|$ iterations. On top of that, $\ell \leq |V| - 1$, since it is a simple path, as such:

$$\text{Total \# of iterations} \leq |E|\,(|V| - 1) = O(|E||V|).$$

Proof. Consider the residual network after T iterations. The length of the shortest $s - t$ path is ℓ. Edges in the graph of BFS from s that go downwards are called the "forward edges". Note that edges can only go down by one level and can connect vertices on the same level or upwards (cannot go more than one level down).

In iteration $T + 1$, we push additional flow to the shortest $s - t$ path, saturating at least

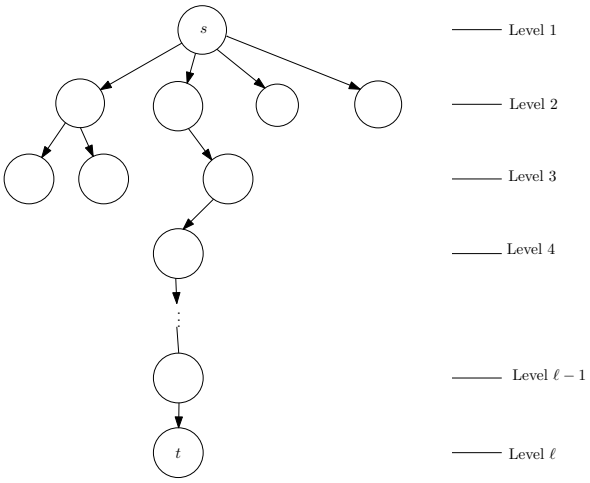


Figure 1

one of the edges on this path. As such the residual network will look as follows:

- All edges on P that are saturated disappear
- we may introduce backward edges connecting to edges in P

Note that it will not fall. Proving that ℓ is increasing. For ℓ to stay the same, you must only use forward edges, otherwise it would increase. Since you remove at least one forward edge during each iteration, you can only stay on the same level for $|E|$ iterations. □

2 March 28th, 2019

2.1 Bipartite Matching

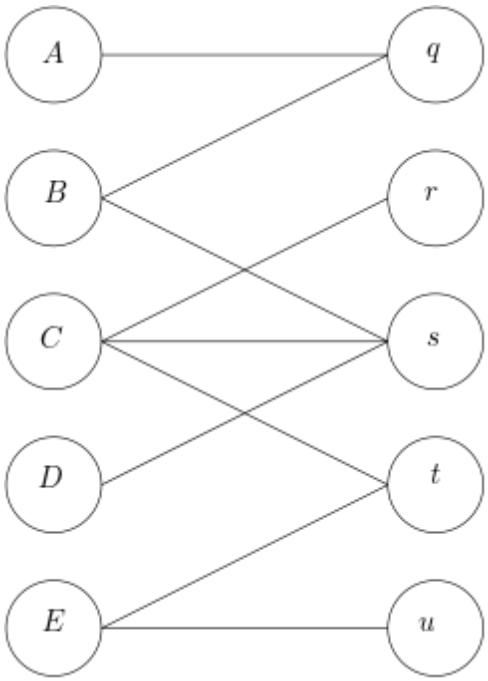


Figure 2: Example of Bipartite Graph

Definition 2.1. A **bipartite graph** is a graph with a bipartition, with all edges only crossing the bipartition.

- An edge represents a person willing to do a job
- Each person can be assigned to at most one job
- Each job can be occupied by at most one person
- QUESTION: is it possible to assign the employees such that every employee gets one job and each job is filled?

Remark 2.2 — A **maximal matching** is a matching for which if you add another edge, it would not be a matching. A **maximum matching** is the matching with maximum cardinality. A maximum matching is always a maximal, but a maximal is not always a maximum.

Definition 2.3. A **neighborhood** of a subset of vertices, A, is all the vertices they are connected to a vertex in A by an edge in E. e.g.

$$N(\{A, B, D\}) = \{q, s\}.$$

Remark 2.4 — The following are necessary conditions to have a perfect matching:

- The size of each side in this partition is the same.
- The size of the neighborhood of any subset is greater than or equal to the size of the subset. e.g. in the example above

$$|N(S)| \geq |S|, \ \forall S \subseteq L.$$

These will be proven to be sufficient later

From weak duality, we have that for any graph, the size of the max matching is less than or equal to the size of the min vertex cover.

2.2 Bipartite Matching as a Max Flow Problem

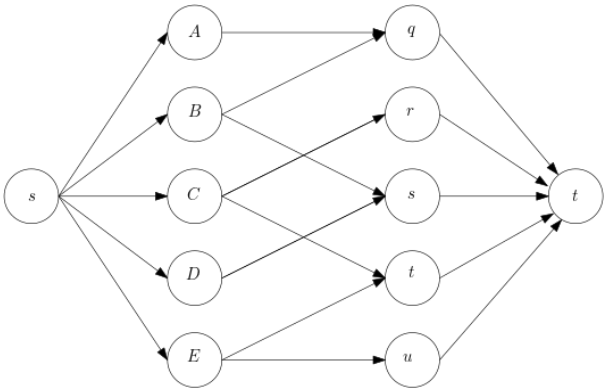


Figure 3: Matching as a Max Flow Problem (all $c(s, v)$ and $c(t, v)$ is 1)

Easy observations:

1. Suppose you have an integral flow of value k in G', the flow network. Then you have a matching of size k in G.
2. Suppose you have a matching os size k is G. Then you have a flow of value k.

Algorithm 2.5

We can solve the problem of finding a max matching in a bipartite graph in polynomial time by reducing it to max flow as follows:

- Let L, R denote the vertices on the left and right side of the bipartite graph respectively. Direct edges of the graph from left to right.
- Add 2 new nodes s and t, and add direct edges from s to all vertices in L and from all vertices in R to t.
- Set $c(s, v) = c(v, t) = 1$ and ∞ for edges from L to R.
- Find the max flow in the network, assuming that the flow is integral.
- Return the edges of the graph in which flow is 1.

This algorithm outputs a maximum matching.

Proof. Follows from (1) and (2) □

Remark 2.6 — Note that we can use any FF method, since the number of iteration is bounded by $|V|$. Since each iteration takes $O(|E|)$ time, we have a $O(|V||E|)$ algorithm.

We can run the FF method to find the maximum matching, as all capacities are integral values (all edges have capacity 1). If we have a flow, we will have a matching equal to $|f|$.

Remark 2.7 — This algorithm only works for bipartite graphs, but there is another polynomial time algorithm that can find the maximum matching of arbitrary graphs (albeit more complicated).

2.3 Perfect Matching in Bipartite Graphs

Definition 2.8. A **perfect matching** is a matching which "covers" all the vertices.

Theorem 2.9 (Hall's Theorem)

A bipartite graph $G(V, E)$ with bipartition (L, R) has a perfect matching if and only if:

1. $|L| = |R|$; and
2. $\forall A \subseteq L$, we have
$$|A| \leq |N(A)|.$$

These properties have been shown to be necessary conditions, but they are also sufficient conditions (using max-flow min-cut).

Proof. Suppose that G does not have a perfect matching and $|L| = |LR|$. Then we will prove that $\exists A \subseteq L$ such that $|A| > |N(A)|$, i.e. (2) will be violated. We will use the same reduction to the max-flow problem. Recall the flow network G' as before.

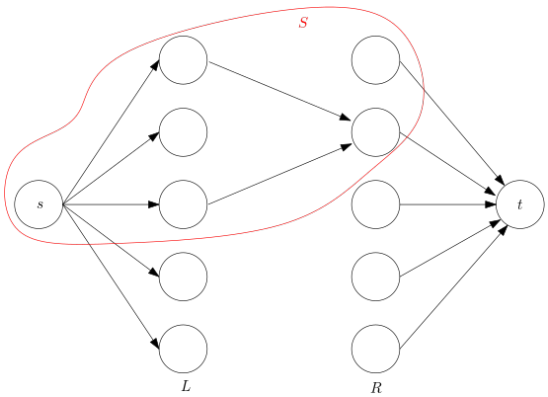


Figure 4: Example of a cut S

Consider the $s-t$ cut S of minimum capacity. The capacity of this cut is

$$|L - S| + |R \cap S|.$$

Note that there is no edges from $L \cap S$ that goes to $R \cap (V - S)$, since the capacity of the cup would be infinite, which would violate the max flow min cut thoerem, since the max flow is clearly finite. This means that

$$N(L \cap S) \subseteq R \cap S.$$

Since G does not have a perfect matching, if we let $|L| = |R| = n$,

$$\text{Size of max matching} < n \implies \text{Value of max flow } < n \implies \text{Capacity of min cut } < n.$$

Note that

$$|L - S| + |R \cap S| < n = |L - S| + |L \cap S|,$$

which means

$$|L \cap S| > |R \cap S|.$$

□

3 April 2nd, 2019

3.1 Matching on Bipartite Graphs Part 2

Theorem 3.1 (Konig's Theorem)
In a bipartite graph, the size of the max matching **equals** the size of the min vertex cover.

Remark 3.2 — This is an example of an exact integrality relaxation.

Proof. Let $C = (R \cap S) \cup (L - S)$.

Claim 3.3. C is a vertex cover.

Claim 3.4. There is no vertex cover of size smaller than $|C|$.

Recall that the capacity of the min cut S is:

$$S = |(L - S)| + |R \cap S| = |C|.$$

By the max-flow min-cut theorem, the value of the max flow is $|C|$, which is also the value of the max matching. From weak duality, we have the size of any vertex cover is greater than the size of the max matching, thus claim 1.4 is proven.

Note that this is a vertex cover because there is no edges from $L \cap S$ to $R - S$, as such, if there were an edge exiting $L \cap S$, then it would edge in $R \cap S$. Similar logic can be applied to $R - S$. In addition, note that direct edges out of s and the direct edges into t are artificial. □

3.2 Linear Program of Max Flow

Remark 3.5 — Note that the min-cut problem is an integer linear program. As such, the dual of the max-flow is not the min-cut (as the max-flow is an LP). Rather the dual is the LP relaxation of the min-cut problem.

We will use the primal dual framework to gain a deeper understanding.

Consider the flow network $G = (V, E)$, with source s, sink t, and capacities c. The LP for the max flow is:

$$\begin{aligned} \text{Maximize:} \quad & \textstyle\sum_v f(s, v) \\ \text{Subject to:} \quad & \textstyle\sum_u f(u, v) = \sum_w f(v, w), \quad \forall v \in V - \{s, t\} \\ & f(u, v) \leq c(u, v), \quad \forall (u, v) \in E \\ & f(u, v) \geq 0, \quad \forall (u, v) \in E \end{aligned}$$

Note that we would have a dual variable for each edge and vertex (which would be difficult). Instead think of the flow as a collection of paths with flows traveling through the path from s to t.

Remark 3.6 — Note that the number of possible paths is exponential, however we will stil consider, as it will give simpler structure (we do not need to worry about flow constraints).

Let us introduce variable x_p, denoting the flows on path p. For each possible $s-t$ path, let P denote the set of all such $s-t$ paths. The LP will be as follows:

$$\begin{aligned} \text{Maximize:} \quad & \textstyle\sum_{p\in P} x_p \\ \text{Subject to:} \quad & \textstyle\sum_{p\in P:\ (u,v)\in p} x_p \le c(u,v), \quad \forall (u,v)\in E \\ & x_p \ge 0, \quad \forall p \in P \end{aligned}$$

Dual: Introduce a variable $y(u,v)$ for each edge (u,v):

$$\begin{aligned} \text{Minimize:} \quad & \textstyle\sum_{(u,v)\in E} c(u,v)y(u,v) \\ \text{Subject to:} \quad & \textstyle\sum_{(u,v)\in p} y(u,v) \ge 1, \quad \forall p \in P \\ & y(u,v) \ge 0, \quad \forall (u,v)\in E \end{aligned}$$

Intepretation: Dual is assigning weights of edges:

- Think of $y(u,v)$ as the "length" of the edge (u,v)
- The constraints say that the length of each path is at least 1 - i.e. distance [length of shortest path] of t from s is at least 1.

Goal: To "separate" s and t while minimizing the total capacity selected.

Remark 3.7 — Note that if the non negativity constraint was integer, then it would be the ILP of the min-cut.

Lemma 3.8

For every feasible cut A in the flow network, there is a feasible solution whose cost is the same as the capacity of A.

Proof. Note that if we assign $y(u,v)$ to 1 for all edges crossing the cut, we would have a feasible solution. □

Lemma 3.9

Given any feasible solution $y(u,v),\ \forall (u,v)\in E$, it is possible to find a cut A such that:

$$c(A) \le \sum_{(u,v)} c(u,v)y(u,v).$$

This means that if we have the optimal solution for the dual, we can find a cut that is just as good.

4 April 4th, 2019

4.1 Fractional Min-cut

Review - LP relaxation of integer LP for min cut

$$\begin{aligned} \text{minimize:} \quad & \textstyle\sum_{(u,v)\in e} c(u,v)y(u,v) \\ \text{subject to:} \quad & \textstyle\sum_{(u,v)\in p} y(u,v) \geq 1, \quad \forall p \in p \\ & y(u,v) \geq 0, \quad \forall (u,v) \in e \end{aligned}$$

$y(u,v)$ can be thought of the length of an edge, with the constraints being that the length of t from s is at least 1.

To prove that the integrality gap to be 1, we need to show that there exists a integer cut that is just as good as the fractional min-cut

Lemma 4.1

Given any feasible solution $y(u,v)$ for $(u,v) \in E$, it is possible to find a $s-t$ cut A such that

$$c(A) \leq \sum c(u,v)y(u,v).$$

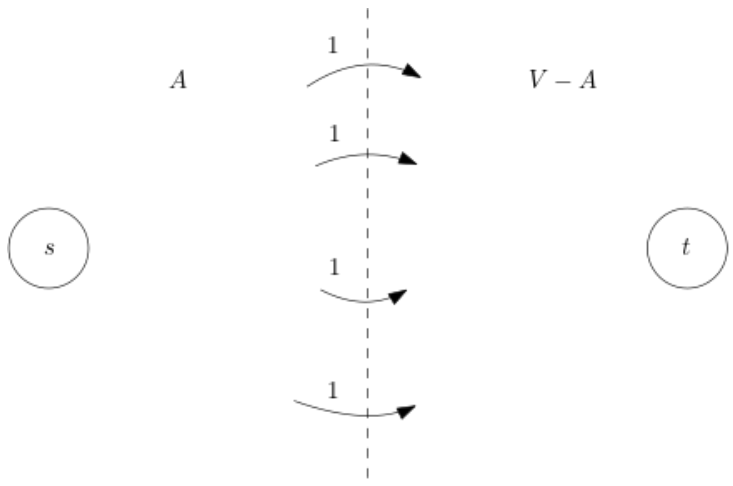
Constraints of LP imply $d(t) \geq 1$.

Pick T (threshold) uniformly at random in interval $[0,1)$. Define A to be the set $A = \{v \ : d(v) \leq T\}$, if 1 was included in the interval, then t could be in A. Now, we will show that

$$E[c(A)] \leq \sum c(u,v)y(u,v).$$

If this is true, then we have proven the above lemma, as there cannot be a cut A which has less capacity then the fractional min-cut. As it's less than or equal to the fractional min-cut, then they must be equal. For this to be consistent, then all edges crossing A are 1, and everything else is 0.

There must be a $s-t$ cut A s.t. this is true, (since it is true on average).



Proof. Define a random variable $X(u,v)$ to be 1 if $u \in A$ and $v \notin A$. Otherwise it is 0.

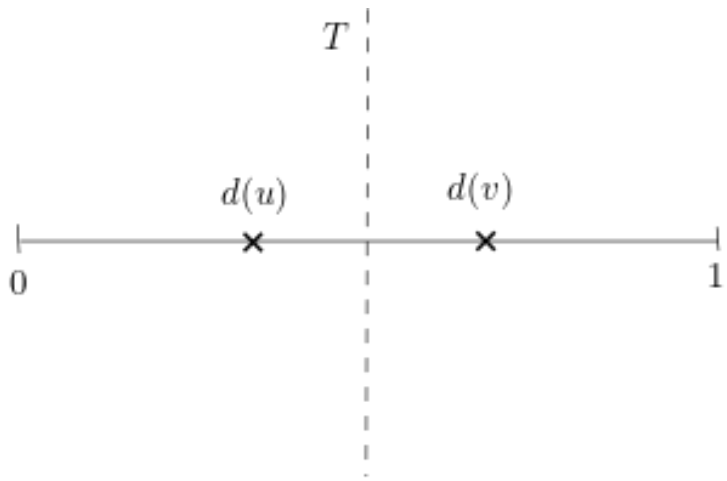
$$c(A) = \sum_{(u,v)\in E} X(u,v)c(u,v)$$

$$E[c(A)] = E[\sum X(u,v)c(u,v).$$

By the linearity of expectation:

$$= \sum c(u,v)E[X(u,v)],$$

$$E[X(u,v)] = \Pr[X(u,v) = 1] = \Pr[d(u) \leq T \text{ and } d(v) > T] \leq d(v) - d(u)$$



$$d(v) \leq d(u) + y(u,v)$$

$$E[X(u,v)] \leq d(v) - d(u) \leq y(u,v).$$

□

4.2 Steiner Forest Problem

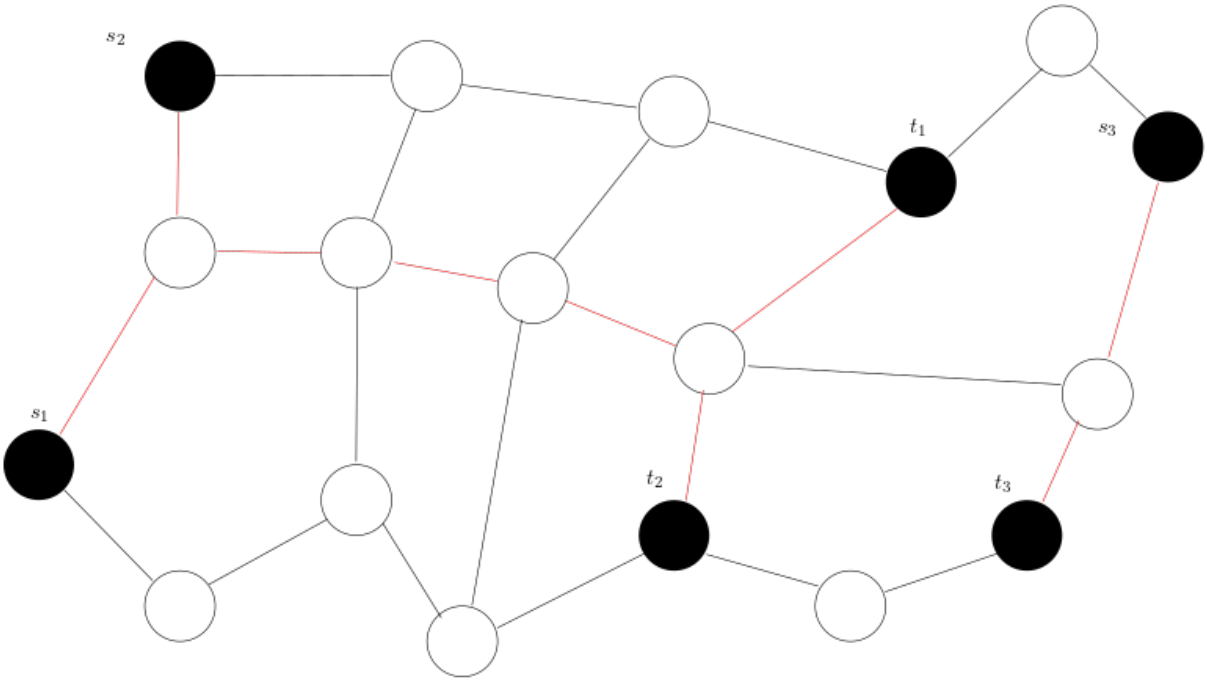
Given a graph $G = (V, E)$, edge cost $C : E \to R^+$. In the Steiner tree problem, we have a set of required vertices that must be connected. In the Steiner Forest problem, we have sets $S_i \subseteq V$, find a minimum cost subgraph F (for forest) such that each pair of vertices belonging to the same set S_i is connected.

Problem Restatement Define a connectivity requirement function r (for requirement) that maps unordered pairs of vertices to $\{0, 1\}$:

$$r(u,v) = \begin{cases} 1, & \text{if } u \text{ and } \text{ belong to some set } S_i \\ 0, & \text{otherwise} \end{cases}$$

Note that there cannot be cycles (can remove an edge to reduce the cost)

	minimize:	$\sum_{e\in E} c_e x_e$	
The ILP is:	subject to:	...	but what are the constraints
		$x_e \in \{0,1\}$	



Note that if $r(u, v) = 1$, for every $u - v$ cut, there must be an edge crossing such cut. This is a necessary condition, but is it sufficient? It is, as if u, v were not connected, there would have been a cut that separates them.

We let $\delta(S)$ denote the set of edges with exactly one endpoint in S. Let $\overline{S} = V - S$. Consider any cut $(S, \overline{S})$ in G that separates a pair (u, v) that should be connected. Then we **must** pick at least one edge $e \in \delta(S)$. Clearly this is necessary, but it is also sufficient.

Let $S^\star$ be the collection of all sets S such that $(S, \overline{S})$ separate a pair (u, v) for which $r(u, v) = 1$. Introduce a 0/1 variable x_e for each edge $e \in E$:

Integer LP for Steiner Forest:

$$\begin{array}{rll} \text{minimize:} & \sum_{e \in E} c_e x_e & \\ \text{subject to:} & \sum_{e : e \in \delta(S)} x_e \geq 1 & \forall S \in S^\star \\ & x_e \in \{0, 1\} & \end{array}$$

This is because each $S \in S^\star$ is a cut that must be crossed.

5 April 9th, 2019

Review

Steiner Forest problem Given a graph $G = (v, E)$ with positive edge weights c_e for each edge $e \in E$ and connectivity requirements

$$r(u,v) = \begin{cases} 1 & \text{if } u, v \text{ must be connected} \\ 0 & \text{otherwise} \end{cases}.$$

- $S^\star$: Collection of subsets $S \subseteq V$ such that for some (u, v) for which some $r(u, v) = 1$, for some $u \in S$ and $v \notin S$.
- $\delta(S)$: Set of all edges crossing cut $(S, \overline{S}$.

5.1 Primal Dual for Steiner Forest

As an LP, this problem is:

$$\begin{aligned} \text{minimize:} \quad & \sum_e c_e x_e \\ \text{subject to:} \quad & \sum_{e:e\in\delta(S)} x_e \geq 1, \quad \forall S \in S^\star \\ & x_e \in \{0, 1\} \end{aligned}$$

Dual LP:

$$\begin{aligned} \text{maximize} \quad & \sum_{S\in S^\star} y_s \\ \text{subject to:} \quad & \sum_{S\in S^\star : e\in\delta(S)} y_s \leq c_e, \quad \forall e \in E \\ & y_s \geq 0 \end{aligned}$$

In words, we are raising the dual of each set $S^\star$ while making sure that no edge leaving it is becoming over-tight.

The number of dual dual variables is equal to the number of cuts for which an edge must cross the cut, which could be exponential. However, many y_s will remain 0, so it will still run in polynomial.

Consider the following:

- Set $y_{\{t_1\}} = 1$, picking all the edges leaving t_1.
- Doing this, all sets in $S^\star$ will be satisfied, as they all have some (t_1, v) crossing the cut.
- The solution has 5, but in general it might have k (for K_k). However, the lower bound is only 1, making the lower bound useless.

The break through idea is to raise the dual variable simultaneously. We can do this beaus any dual feasible solution is a valid lower bound, so we can grow the duals in any way. Note that for this example, we can raise all dual variables to 0.5, meaning our lower bound is $\frac{k+1}{2}$. (Note that we are only concerned with the y_S for single term sets, with all others still 0).

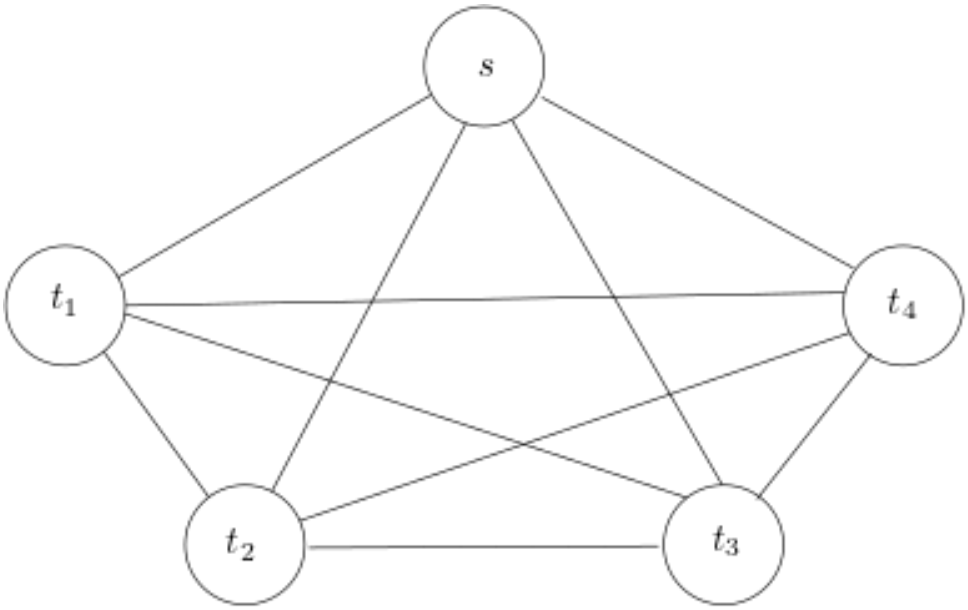


Figure 5: Complete graph with all edges with weight 1. Note that all subsets are in $S^\star$

5.2 Approximation Algorithm for the Steiner Forest Problem

Remark 5.1 — **NEW IDEA**: Raise duals in a synchronized manner. We are not trying to satisfy a singly unsatisfied primal constraint, but we are trying out many possibilities at the same time.

This might seem ad hoc at first, but it turns out that it could be applied to many situations. Instead of raising one dual at a time, raise them all together.

Remark 5.2 — Terminology:

- We say that edge e **feels** dual y_s if $y_s > 0$ and $e \in \delta(S)$.
- We say that edge e is **tight** if the total amount of dual it feels equals its cost, i.e.
$$\sum_{S \in S^\star : e \in \delta(S)} y_e = c_e.$$
- We say that edge s is **over-tight** if the total amount of dual it feels exceeds its cost.
- We say that a set S has been **raised** if $y_S > 0$

Theorem 5.3
The dual is trying to maximize the sum of dual variables y_S subject to the condition that no edge feels more dual than its cost - i.e. no edge is over-tight.

Let us consider how out algorithm will work with this example.

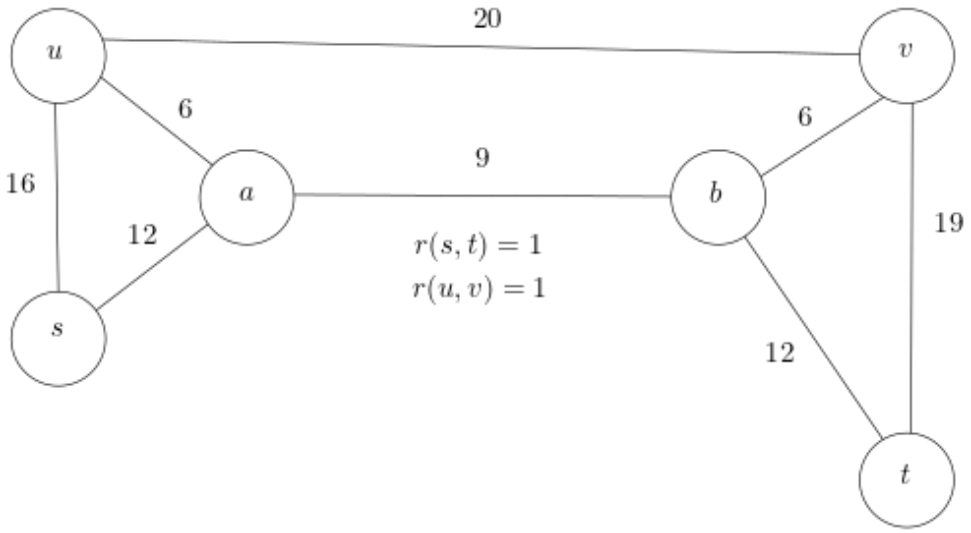


Figure 6: Optimal Solution: $(u,a),(s,a),(a,b),(b,t),(b,c)$

- Let us consider the single vertex cuts of $\{u\},\{s\},\{v\},\{t\}$. Note that $\{a\},\{b\} \notin S^\star$. We call a, b **inactive components**
- We can raise all of these dual by 6, which will make $(u,a),(b,v)$ tight.
- If we follow the logic in the past, we might create a cycle, so for this example, we pick (u,a).
- This means that $\{s\},\{u,a\},\{v\},\{t\}$ are the new **active** components since they are in $S^\star$.
- Now we raise the duals of the new active components, note that $\{u\}$ is inactive, but $\{u,a\}$ is active.
- We do this by going through all edges and finding the most serious constraint. Since (v,b) is already tight, we raise them by 0 and pick (v,b).

> **Remark 5.4** — For each iteration, we pick exactly one edge, and we are allowed to raise the duals by 0.

- We raise $\{s\},\{u,a\},\{v,b\},\{t\}$ by 2, which makes (s,u) tight, and so we pick it.
- Now we have three active components $\{u,a,s\},\{v,b\},\{t\}$, and we could raise them by 1, making (b,t) tight.

> **Remark 5.5** — Make sure you keep track if the edge is adjacent to 1 or 2 active components.

- Now we have two active components $\{u,a,s\},\{b,v,t\}$. We can raise them by 1 each, making (u,v) tight.
- After this, we have 1 component which is inactive.

Remark 5.6 — After doing this, we need to check if every edge is needed, as some are redundant, e.g. (u, a). (Special pruning step).

- The solution we return is $16 + 20 + 6 + 12 = 44$.
- The lower bound is $6 \times 4 + 2 \times 4 + 1 \times 3 + 1 \times 2 = 37$.

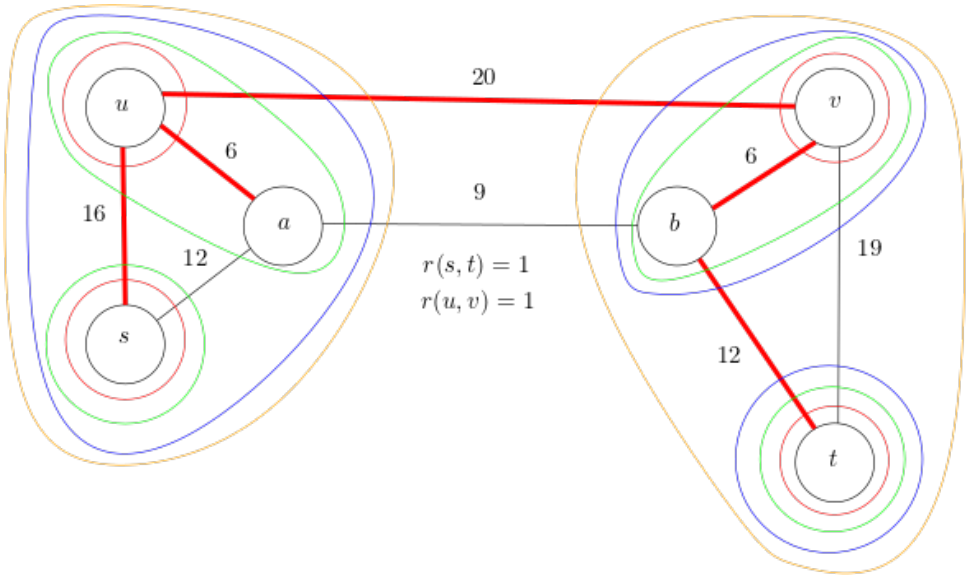


Figure 7: Example of the Algorithm

Algorithm 5.7

- Say the algorithm has picked some edges so far, which forms a forest F.
- We say that a set S is **unsatisfied** if $S \in S^\star$ but there is no picked edge crossing the cut $(S, \overline{S})$.
- Clearly if F is not primal feasible, then there is a connected component in F that is unsatisfied. We say that this component is **active**.
- In each iteration, we raise the dual of each active component until some edge goes tight. We pick one of the tight edges, and repeat.
- We stop when all connectivity requirements are satisfied, i.e. no sets are unsatisfied.
- Finally do the pruning step by removing the redundant edges.

Index