COMP5712 - Combinatorial Optimization Spring 2019

Taught by Sunil Arya

Notes by Aaron Wang

These are the notes that I typed in class starting from after midterm. There's probably a lot of typo/mistakes since I haven't really gone through them after class, so keep an eye out for anything that doesn't make sense.

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

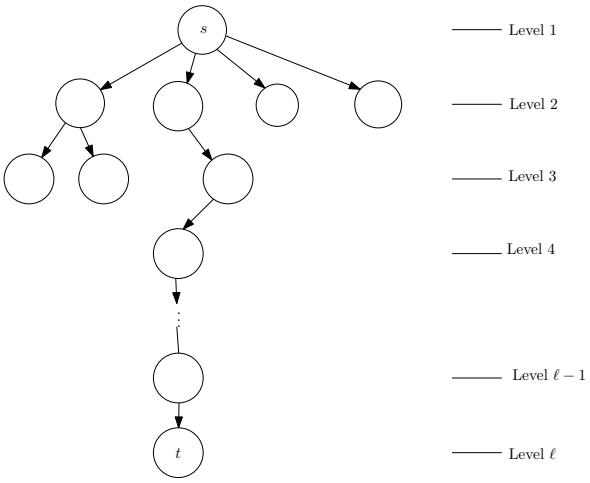


Figure 1

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

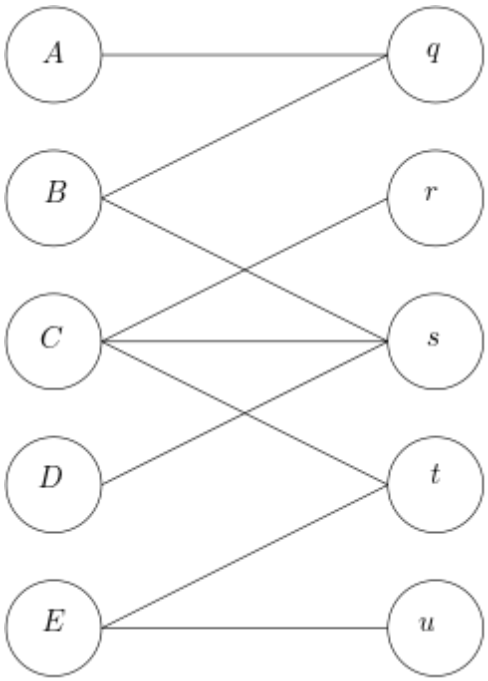


Figure 2: Example of Bipartite Graph

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

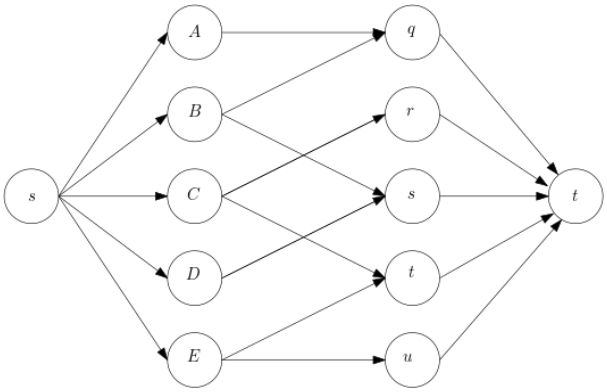


Figure 3: Matching as a Max Flow Problem (all $c(s, v)$ and $c(t, v)$ is 1)

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 $|L| = |R|$. 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.

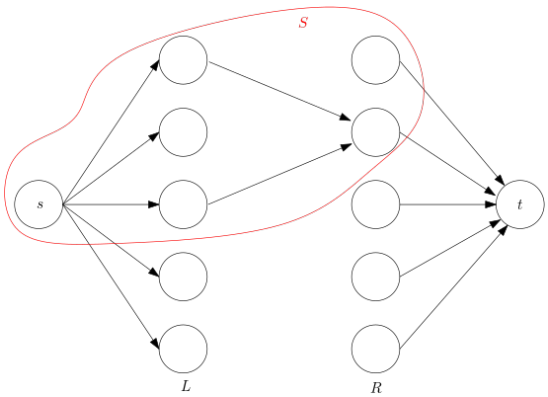


Figure 4: Example of a cut S

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 cut could be infinite (if all edges crossing the bipartition have infinite capacity), 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

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

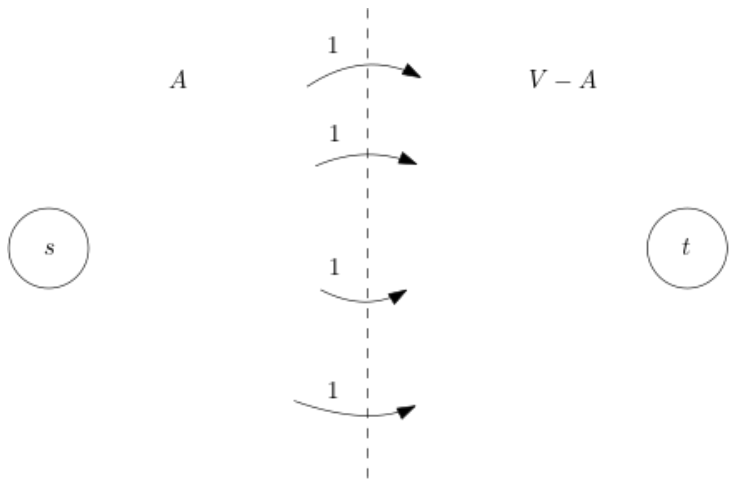
Constraints of LP imply $d(t) \geq 1$.

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.

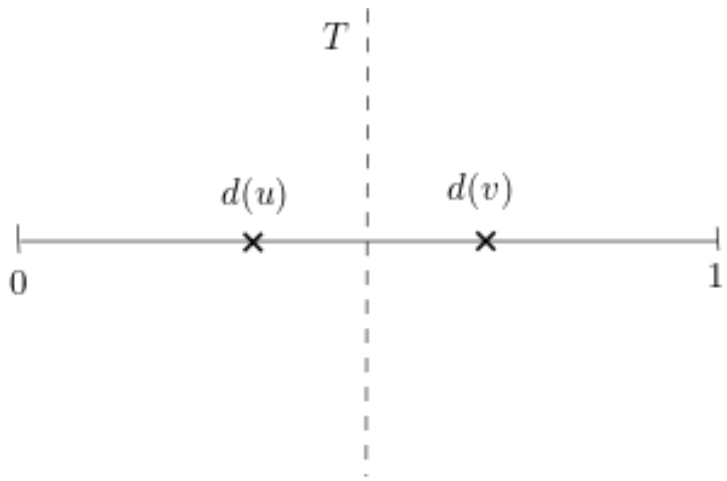
$$c(A) = \sum_{(u,v)\in E} X(u,v)c(u,v)$$

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

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

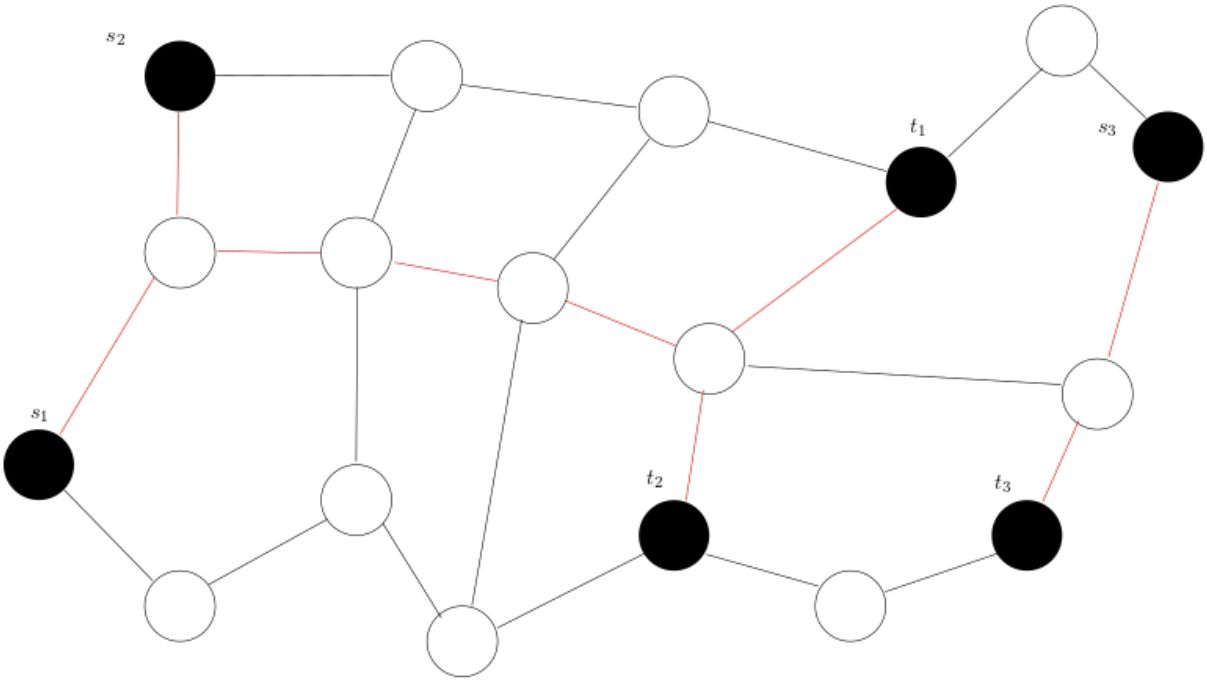
Given a graph $G = (V, E)$, edge cost $C : E \to R^+$. In the Steiner tree problem, we have a set of required vertices that must be connected. In the Steiner Forest problem, we have sets $S_i \subseteq V$, find a minimum cost subgraph F (for forest) such that each pair of vertices belonging to the same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

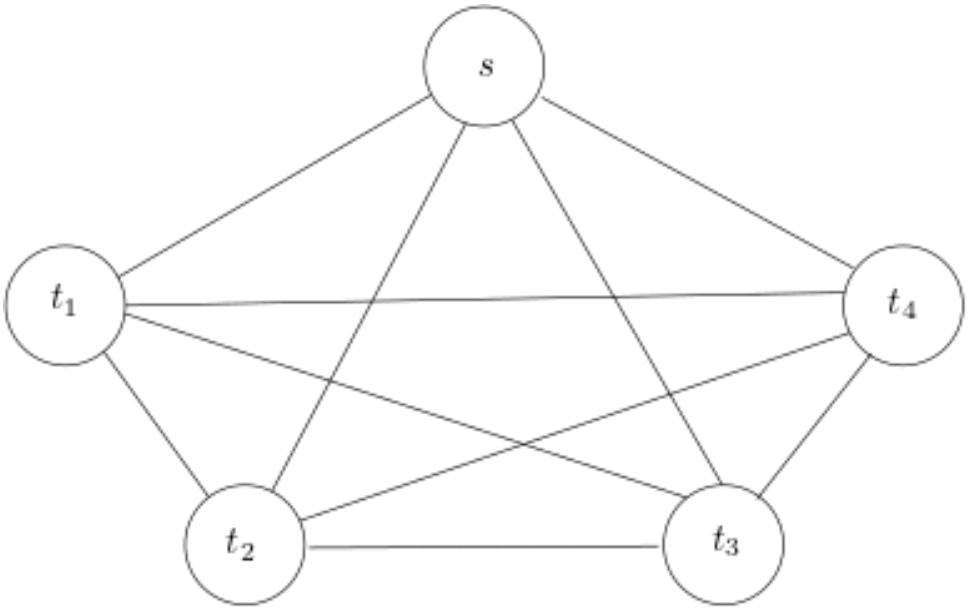


Figure 5: Complete graph with all edges with weight 1. Note that all subsets are in $S^\star$

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.

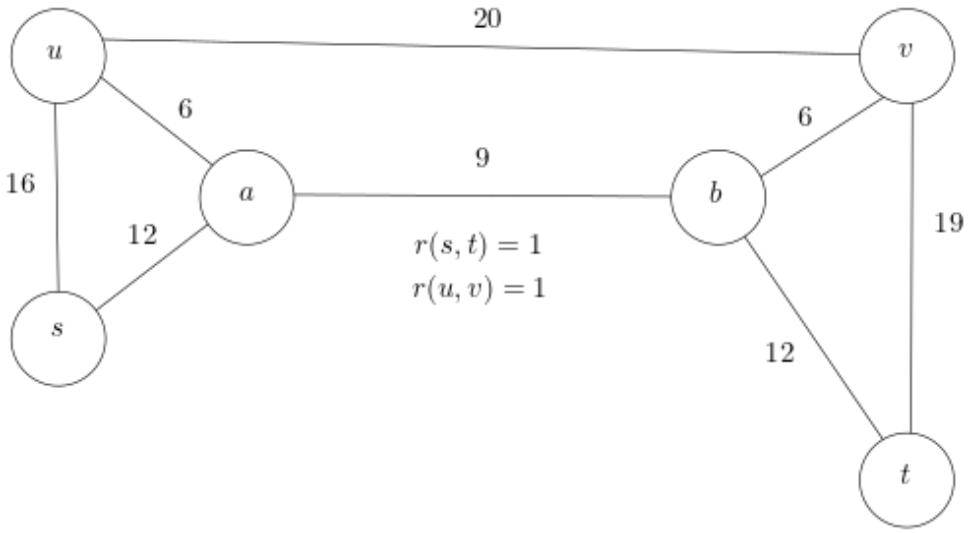


Figure 6: Optimal Solution: $(u, a), (s, a), (a, b), (b, t), (b, c)$

- 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 $6 \times 4 + 2 \times 4 + 1 \times 3 + 1 \times 2 = 37$.

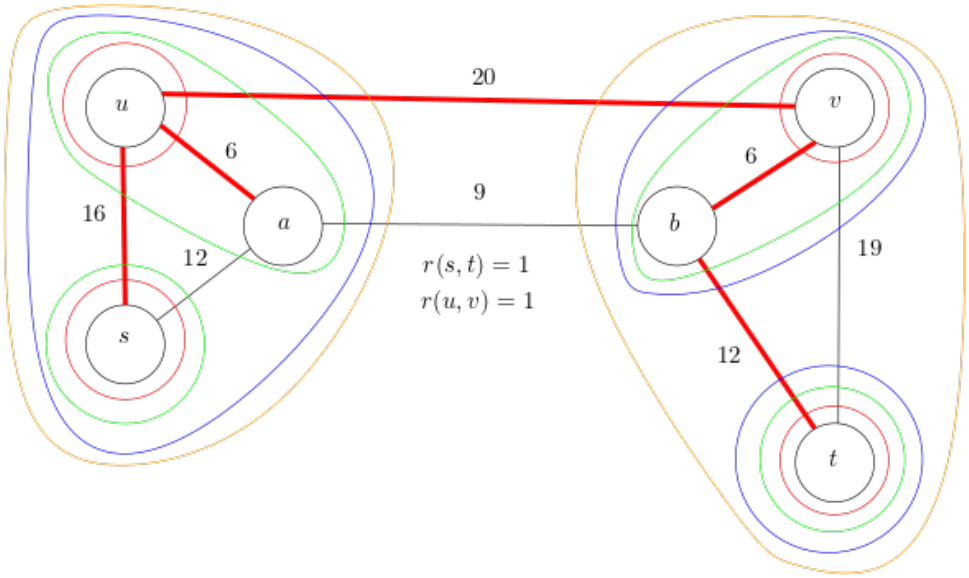


Figure 7: Example of the Algorithm

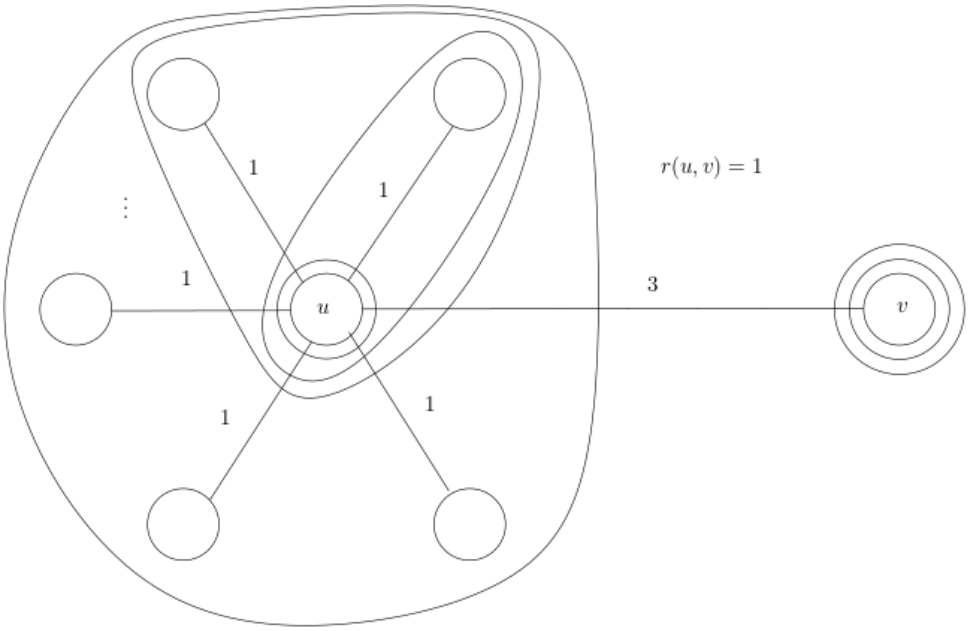
Algorithm 5.7

- 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.

6 April 11th, 2019

Review

- At each iteration, the algorithm picks exactly one edge.
- The edges picked form a forest F.
- Active component are those that belong to $S^\star$.
- If F is not primal feasible, then there must be some connected component in F that is unsatisfied (active).
- We raise the dual simultaneously for all active components in a synchronized manner until some edge goes tight.
- We pick one of the edges and merge such components, terminating the iteration.
- We stop when a primal feasible solution F is formed (no active components).



Dual: $2 + 1 = 3$

Cost of solution: $3 + 1\times\#$ of spokes $= 3 + (|V| - 2 = |V| + 1 = \Omega(|V|)$

- As shown above, the solution might give us a terrible approximation ratio, which is why we have a pruning step to drop the redundant edges. After doing so, we would have cost of solution = 3.
- This can be done by seeing if $F - \{e\}$ satisfy the connectivity constraints, removing it if it is redundant. This will give us F'.

6.1 Steiner Forest Algorithm Correctness and Approx. Ratio

- **Correctness** - At termination, dual is feasible and primal is feasible.

  Proof. – Dual is feasible because as soon as the dual constraint goes tight, the dual variable is frozen and thus the edges do not become over-tight.

  – Primal is feasible because:

  1. Before the pruning step, the primal solution obtained is feasible (we stop only when all constraints of F are satisfied.
  2. The pruning step does not hurt feasibility. Note that for each pair of vertices u, v that have a connectivity constraint ($r(u, v = 1)$, there is a unique path between the two in F. Each edge e on that path will not be removed, as u, v would not be connected.

  □

- **Approximation Ratio** The approximation ratio of this algorithm is 2.

  Proof. – By weak duality,

  $$\sum_{S \in S^\star} y_s \leq OPT,$$

  where OPT is the cost of the optimal Steiner Forest.

  – Note:

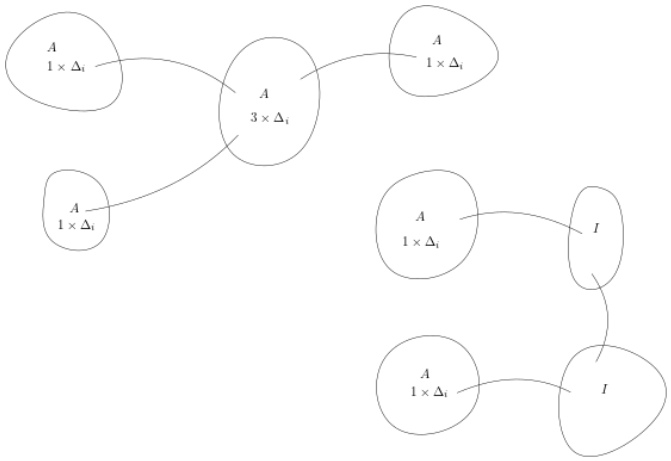
  $$\sum_{S \in S^\star} y_s = \sum_i \left(\Delta_i \times \#\text{ of active components}\right),$$

  where Δ_i is the amount by which the duals of each active components is raised on the i-th iteration. (try to do this without using LP-duality)

  – Now we'd like to show that $Cost(F') \leq 2 \times \sum_{S \in S^\star} y_S \leq 2 \times OPT$.

  – Let us define:

  Degree of $S \subseteq V :=$ # of edges in F' that belong to the cut $(S, \overline{S})$.



- The above diagram demonstrates that:

$$Cost(F') = \sum_i \left(\Delta_i \times \sum_{S \text{ is active}} \text{Degree of } S \right).$$

- Suppose that Degree of $S \leq 2$, we would have our approximation ratio. (however this is not true, as shown above).
- Consider the average:

$$Cost(F') = \sum_i \left(\Delta_i \times \sum_{S \text{ is active}} \text{Degree of } S \right)$$

$$= \sum_i \left(\Delta_i \times \#\text{ of active components} \times \text{Average Degree of active components} \right).$$

  meaning we have to show that the average degree of active components is less than or equal to 2.
- For a tree, it has n vertices and $n-1$ edges, meaning that average degree $= \frac{2(n-1)}{2} \leq 2$. So this is true for a forest.
- As such, if all components are active, then we are done, however this is not true in general, as an active component might have many inactive components connected.
- However, due to the pruning step, all inactive components must be internal.
- Consider the simple case first. Suppose there were no inactive components in iteration i. The average degree of the active components $\leq$ average degree of a tree ≤ 2.
- If there are inactive components that are leaves, we might be in trouble, as they would have degree 1, meaning that the average degree of active components could be ≥ 2.
- However, the clearly cannot be the case because of the pruning step.
- Thus any inactive component must have degree ≥ 2 as they are internal, meaning that the degree of active components ≤ 2.

Remark 6.1 — What if an inactive component has degree 0? (only consider the connected components with active component) TODO: think about this more

□

7 April 16th, 2019

7.1 Multiway Cut Problem

- Given an undirected graph $G = (V, E)$, with costs $c_e \geq 0$ for all edges $e \in E$ and k distinguished vertices $s_1, s_2, \ldots, s_k$
- Remove a minimum cost set of edges F such that no pair of distinguished vertices are in the same connected components of $(V, E - F)$.

Remark 7.1 — For $k = 2$, we can reduce this problem into the min $s-t$ problem, by duplicating all undirected edges to get a directed graph. As such for $k = 2$, there is a polynomial time solution that can solve this problem exactly. However, for $k \geq 3$, this problem is NP-Complete.

Application in distributed computing:

- Vertices represent "Objects".
- c_e represents amount of communication between objects.
- We need to place the "Objects" in k different machines with a special object s_i which needs to be on machine i.
- Goal is to partition all the objects onto the k machines so as to minimize the cost of communication.

7.2 2-Approximation Algorithm

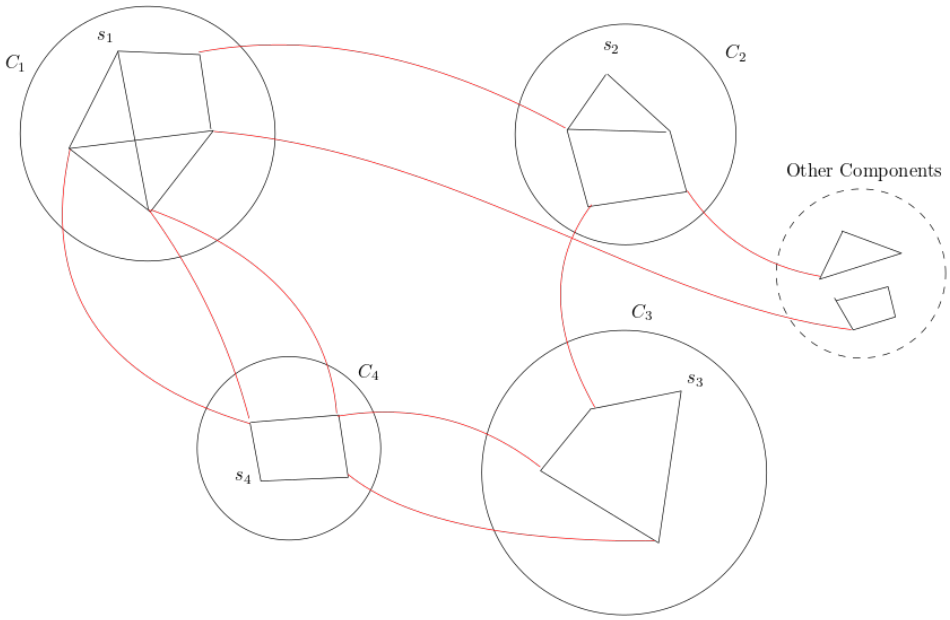


Figure 8: F is the set of red lines.

- Define $F_i = \delta(C_i)$ to be the edges in the cut $(C_i, \overline{C_i})$.
- F : feasible solution for the multiway cut.
- Note that F_i is a cut separating s_i from all the other distinguished vertices. Call F_i an **isolating** cut as it isolates s_i from all the other distinguished vertices. As such:

$$F = \bigcup_i F_i,$$

and

$$\sum_i Cost(F_i) \leq 2\ Cost(F).$$

This is because each edge is counted for at most twice (note that edges to "Other Components" are only counted once.

- Note that these above properties apply to any feasible solution F, in particular the optimal solution.

Remark 7.2 — Let F_i be the minimum isolating cut for s_i (assuming this can be found in polynomial time). $F = \bigcup_i F_i$ is the output of our algorithm. Note that

$$Cost(F) \leq \sum Cost(F_i) \leq \sum Cost(F_i^*) \leq 2\ Cost(F^*).$$

Now we just need to find the minimum isolating cut.

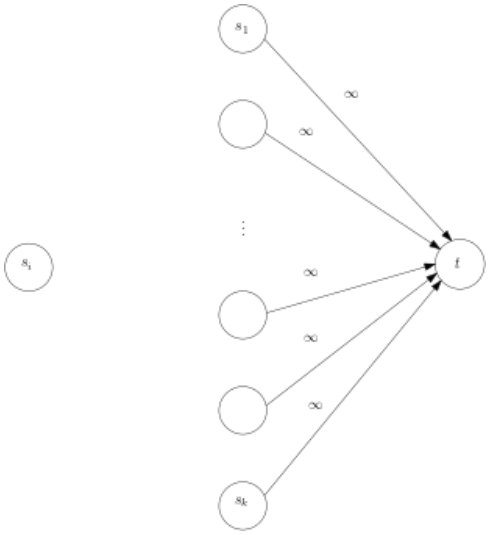


Figure 9: F_i can be solved exactly using the $s-t$ min cut in poly-time

Remark 7.3 — Note that we can drop the most expensive isolating cut, as if all of the others are isolated, then the last one must be. This will improve our solution by $\left(1 - \frac{1}{k}\right)$, since the most expensive isolating cut must have cost greater or equal to $\frac{1}{k}\ Cost(F^*)$.

This means that the new approximation ratio is $2\left(1-\frac{1}{k}\right)$, which means that $k=3$ has approximation ratio $\frac{4}{3}$, $k=4 \implies \frac{3}{2}$, with $k \to \infty \implies 2$.

We will now find an approximation algorithm with approximation factor 1.5.

7.3 1.5-Approximation Algorithm

Reformulation of the problem:

- Partition the set of vertices into sets C_i such that $s_i \in C_i$ for all i and such that the cost of
$$F = \bigcup_i \delta(C_i) \text{ is minimized,}$$
where $\delta(C_i)$ is the set of edges in the cut $\left(C_i, \overline{C_i}\right)$.
- Note that this means that the vertices in the "Other Components" can be placed in any group C_i.
- For each vertex $u \in V$, introduce k different variables, $x_u^1, x_u^2, \ldots, x_u^k$, with
$$x_u^i = \begin{cases} 1 & \text{if } u \text{ is in set } C_i \\ 0 & \text{otherwise} \end{cases}.$$
Thus our goal is:
$$\min \sum_{e=(u,v)} C_e \left[\frac{1}{2}\sum_i \left|x_u^i - x_v^i\right|\right]$$

Remark 7.4 — Note that $\frac{1}{2}\sum_i |x_u^i - x_v^i| = 1 \iff e \in \delta(C_i)$.

However, this is not a valid objective function.

- To fix this, introduce variables z_e^i for each edge $e \in E$ with"
$$z_e^i = \begin{cases} 1 & \text{if } e \in \delta(C_i) \\ 0 & \text{otherwise} \end{cases}.$$
- This allows us to rewrite our objective function as:
$$\min \sum_{e=(u,v)} C_e \left[\frac{1}{2}\sum_i z_e^i\right].$$
The constraints are:
$$\sum_i x_u^i = 1, \quad \text{for all } u \in V$$
$$z_e^i = |x_u^i - x_v^i|.$$
However, the second set of constraints are invalid. To fix this, we have:
$$z_e^i \geq x_u^i - x_v^i$$
$$z_e^i \geq x_v^i - x_u^i,$$
for all edges $e = (u, v)$.

Remark 7.5 — Note that this will make $z_e^i = 1 \iff e \in \delta(C_i)$.

- We also have the connectivity and integer constraints:

$$x_{s_i}^i = 1$$

$$x_u^i \in \{0, 1\}.$$

8 April 25th, 2019

8.1 Review of Multiway Cut Problem

We are given an undirected graph $G = (V, E)$ with edge costs $c_e \geq 0$ for all $e \in E$. We also have k distinguished vertices $s_1, s_2, \ldots, s_k$.

Our goal is to partition the set of vertices into sets C_i such that $s_i \in C_i$ for all i and such that the cost of

$$F = \bigcup_i \delta(C_i)$$

is minimum. In other words, we are trying to minimize the cost of the edges that cross the cuts.

This can be formulated as an ILP:

- For each vertex $u \in V$, introduce variable x_u^i defined by:

$$x_u^i = \begin{cases} 1 & \text{if } u \text{ is assigned to } C_i \\ 0 & \text{otherwise} \end{cases}.$$

- For each edge $e \in E$, introduce variables z_e^i which is defined by:

$$z_e^i = \begin{cases} 1 & \text{if } e \in \delta(C_i) \\ 0 & \text{otherwise} \end{cases}.$$

- ILP:

$$\min_e c_e \left(\frac{1}{2} \sum_i z_e^i \right)$$

  Subject to:

$$\sum_i x_u^i = 1$$

$$z_e^i \geq x_u^i - x_v^i$$

$$z_e^i \geq x_v^i - x_u^i$$

  for all edges $e = (u, v)$, with connectivity and integer constraints:

$$x_{s_i}^i = 1$$

$$x_u^i \in \{0, 1\}.$$

- For the LP relaxation, we have:

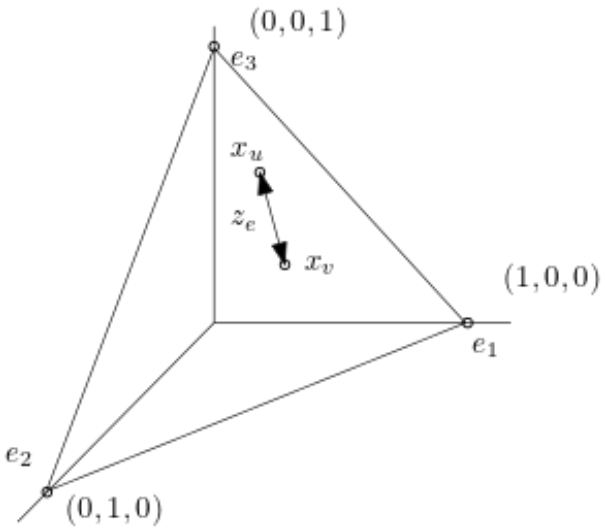
$$x_u^i \geq 0.$$

8.2 LP Relaxation for the Multiway Cut Problem

Think of x_u as a point in k dimensions:

$$x_u = (x_u^1, x_u^2, \ldots, x_u^k).$$

For $k = 3$ we can think as the following diagram:



Remark 8.1 — Note that $x_{s_1} = (1,0,0), x_{s_2} = (0,1,0)$, i.e.

$$x_{s_i} = e_i.$$

This means that x_u is on the hyperplane going through e_i. In addition, note that the x_u for the ILP lie on e_i.

Define the simplex in k dimension as:

$$\Delta_k = \{x \in R^k : \sum_i x^i = 1\}.$$

Note that:

$$\sum_i z_e^i = sum_i |x_u^i - x_v^i|$$

$$\|x_u - x_v\|_1,$$

which is the L_1 distance between x_u and x_v.

We can rewrite the relaxed LP as:

$$\min \frac{1}{2} \sum_{e=(u,v)} c_e \|x_u - x_v\|$$

subject to:

$$x_u \in \Delta_k$$

$$x_{s_i} = e_i.$$

In other words, we have to map the vertices the interior of this triangle, with cost being the weighted sum of the endpoints. This is called the **fractional multiway cut**

Remark 8.2 — Although this does not look like an LP, since it is the same as the LP, we can solve this in polynomial time.

Now we will convert this back into a feasible solution through randomized rounding.

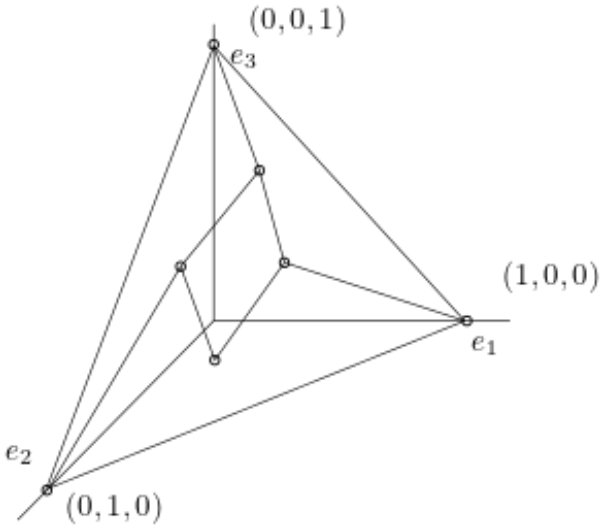


Figure 10: Fractional Mutiway Cut

8.3 Randomized Rounding

The deterministic approach would be to take each vertex and map it to its nearest corner. However, there isn't much analysis to prove its factor. As such we will take a randomized approach.

If we call the "ball distance" as half the L_1 distance, the maximum ball distance between any two vertices is 1 (if they are at the corners). Define $B(i_i, r)$ be the set of vertices u such that:

$$\frac{1}{2}\|e_i - x_u\| \leq r.$$

Note that all vertices are within ball distance 1 from s_i, that is $B(e_i, 1) = V$. (note that we use e_i and s_i interchangeably.

Algorithm 8.3
- Solve LP optimally
- Select $r \in (0, 1)$ uniformly at random
- Assign vertices within $B(s_i, r)$ to C_i

Remark 8.4 — The above algorithm has the following issues:

1. Some vertices are not within ball-distance r from any corner.
2. Some vertices for which there is a conflict, where it is being assigned to multiple corners.

Algorithm 8.5
- Select $r \in (0, 1)$ uniformly at random.
- Select a random permutation Π of $\{1, 2, \ldots, k\}$
- Examine the vertices in the order given by the permutation Π.
- For index $\Pi(i)$, assign all vertices not assigned so far in $B(s_{\Pi(i)}, r)$ to $C_{\Pi(i)}$
- At the end of the order (i.e. when considering $S_{\Pi(i)}$), assign all vertices not yet assigned to $C_{\Pi(k)}$.

Remark 8.6 — Note that both issues with the previous algorithms are solved, as each vertex is given a chance to grab vertices, and all vertices are assigned.

8.4 Analysis of Randomized Rounding Algorithm

To analyze this algorithm, we need to find the probability that edge $e = (u, v)$ makes a contribution to the cost, i.e. u and v are assigned to different corner (if they are assigned to the same, then they would not make a contribution).

First assume that the probability that we separate u and v is $\frac{1}{2}\|x_u - x_v\|$. If we can show this, then the expected cost of the multiway cut will be as good as the fractional multiway cut. (which is not possible). As such, to have a approximation factor of 1.5, we would like to show that the probability is at most $\frac{3}{4}\|x_u - x_v\|$, as:

$$\text{Expected Cost} = \sum_{e=(u,v)} \frac{3}{4} c_e \|x_u - x_v\| = \frac{3}{2} \sum_{e=(u,v)} \frac{1}{2} \|x_u - x_v\|.$$

This is where we are headed.

(reworded) Recall that:

$$OPT \geq \frac{1}{2} \sum_{e=(u,v)} c_e \|x_u - x_v\|$$

where x_u and x_v correspond to vertices u, v respectively in the optimal LP solution. Suppose we can show that the probability that u and v are assigned to different C_i's is $\leq \frac{3}{4}\|x_u - x_v\|$, then the expected solution found has cost:

$$\leq \sum_{e=(u,v)} \frac{3}{4} c_e \|x_u - x_v\| = \frac{3}{2} \sum_{e=(u,v)} \frac{1}{2} \|x_u - x_v\| \leq \frac{3}{2} OPT.$$

We will show that this is the case.

9 April 30th, 2019

9.1 Randomized Multiway Cut Problem

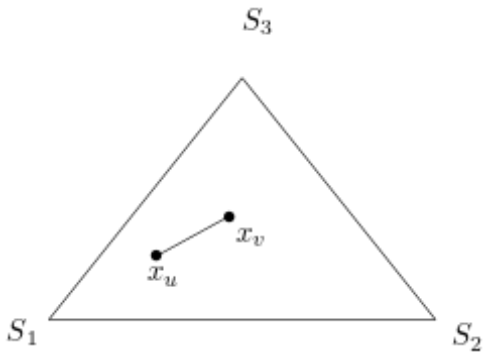


Figure 11: For $k = 3$

- Select $r \in (0, 1)$ uniformly at random
- Consider "corners" of simplex in random order
- Each corner considered "grabs" all vertices within "ball-distance" r from it that have not yet been grabbed. i.e.

$$u \in B(S, r).$$

- The last corner considered grabs all remaining vertices.
- To achieve an approximation factor of 1.5, we need to show that the probability that u and v are assigned to different corners is $\leq \frac{3}{4\|x_u - x_v\|}$.

Lemma 9.1
$u \in B(S_i, r)$ if and only if $1 - x_u^i \leq r$.

Proof. Ball distance of x_u from $S_i = \frac{1}{2}\|x_u - S_i\|$

$$= \frac{1}{2}\left[(1 - x_u^i) + \sum_{j \neq i} x_u^j\right] = \frac{1}{2}\left[(1 - x_u^i) + (1 - x_u^i\right] = 1 - x_u^i.$$

This is because the j-th component S_i is 0 for all $j \neq i$ and 1 for $j = i$, and $\sum x_u^j = 1$. □

Definition 9.2. We say that an index i **cuts** (u, v) if exactly one of x_u and x_v is contained in ball $B(S_i, r)$.

For x_u and x_v to be assigned in different corner, it if necessary that there is some index that cuts (u, v). This is not sufficient since u and v could both be grabbed by a vertex, but another vertex later on might cut the edge (since they are selected in a random order).

When does index i cut (u, v)? In fact, what is the probability that index i cuts (u, v). From the lemma above, we only need to look at that i-th coordinate of x_u^i.

WLOG, assume that x_u is closer to S_i. Then

$$(u, v) \text{ is cut by index } i \text{ iff } r \in (1 - x_u^i, 1 - x_v^i).$$

$$\implies \Pr(\text{index } i \text{ cuts } (u, v)) = \frac{(1 - x_i^v) - (1 - x_i^u)}{1} = |x_u^i - x_v^i|.$$

Now we have:

$$\Pr((u, v) \in \text{multicut}) = \Pr(u \ \& \ v \text{ are cut by index } i)$$

$$\leq \sum_i \Pr(u \ \& \ v \text{ are cut by index } i \ ) = \sum_i |x_u^i - x_v^i| = \|x_u - x_v\|.$$

Now we would like to tighten the restriction to achieve an upper bound of $\frac{3}{4}\|x_u - x_v\|$. Note that we have not yet used the fact that "corners" are considered in random order.

Definition 9.3. We day that an index i **settles** (u, v) if i is the first index in the random permutation such that at least one of u or v belongs to $B(S_i, r)$.

Intuitively, i decides whether (u, v) will be in the multiway cut or not. Let us define the following events:

- S_i : Event that index i settles (u, v).
- X_i : Event that index i settles (u, v).

Then:

$$\Pr\left[(u, v) \in \text{ multicut}\right] \leq \sum_i \Pr(S_i \wedge X_i).$$

This is because (u, v) is in the multiway cut only if there is some index i that both settles and cuts (u, v), i.e. it is the first index that separates u and v. This is clearly necessary, but it is not sufficient, since it could be the last index.

Let l be the index such that S_i is closest to one of the two endpoints of (u, v). That is, $i = l$ minimizes the quantity $\min(\|S_i - u\|, \|S_i - v\|)$

Claim 9.4. Index $i \neq l$ cannot settle edge (u, v) if l is ordered before i in the random permutation.

Proof. If index i were to settle (u, v), then it would be l. □

We need to compute $\Pr(S_i \wedge X_i)$ for index i. There are 2 cases: $i \neq l$ and $i = l$.

1. For $i \neq l$, we have:

$$\Pr\left[S_i \wedge X_i\right]$$

$$= \Pr\left[S_i \wedge X_i \mid l \text{ occurs after } i \text{ in permutation}\right] \times \Pr[l \text{ occurs after } i \text{ in permutation}]$$

$$+ \Pr\left[S_i \wedge X_i \mid l \text{ occurs before } i \text{ in permutation}\right] \times \Pr\left[l \text{ occurs before } i \text{ in permutation}\right]$$

$$= \frac{1}{2}\Pr[S_i \wedge X_i \mid l \text{ occurs after } i \text{ in permutation}] \leq \frac{1}{2}\Pr[X_i] = \frac{1}{2}|x_u^i - x_v^i|.$$

2. For $i = l$ we have:
$$\Pr[S_l \wedge X_l] \leq \Pr[X_l] = |x_u^l - x_v^l|.$$

This means that:

$$\Pr\left[(u,v) \in \text{ multicut}\right] \leq \sum_i \Pr(S_i \wedge X_i) \leq \frac{1}{2}\sum_{i \neq l} |x_u^i - x_v^i| + |x_u^l - x_v^l|.$$

Lemma 9.5

For any index l and any 2 vertices u and v,

$$|x_u^l - x_v^l| \leq \frac{1}{2}\|x_u - x_v\|.$$

Proof. $|x_u^l - x_v^l| = \left|\left(1 - \sum_{j \neq l} x_u^j\right) - \left(1 - \sum_{j \neq l} x_v^l\right)\right| = \left|\sum_{j \neq l}(x_v^j - x_u^j)\right| \leq \sum_{i \neq j} |x_v^j - x_u^j| + \|x_v^l - x_u^l\|$ □

$$\Pr\left[(u,v) \in \text{ multicut}\right] \leq \frac{1}{2}\sum_i |x_u^i - x_v^i| + \frac{1}{2}|x_u^l - x_v^l| \leq \frac{1}{2}\|x_u - x_v\| + \frac{1}{2} \times \frac{1}{2}\|x_u - x_v\| = \frac{3}{4}\|x_u - x_v\|.$$

10 May 7th, 2019

10.1 Well-Characterized Problems

Definition 10.1. **NP**: Class of decision problems for which there is a "yes-certificate" (i.e. there for yes-instances, there is a "short" proof that the answer is yes)

Note that this is not symmetric, as such, we have the CO-NP class

Definition 10.2. **CO-NP**: Class of decision problems for which there is a "no-certificate"

We know that:

$$\text{Hamiltonian Cycle Problem} \in \text{NP}.$$

However, we still don't know if the Hamiltonian Cycle Problem is in CO-NP. The belief among experts is that:

$$\text{Hamiltonian Cycle Problem} \in \text{CO-NP}.$$

Note that $\text{P} \subset \text{CO-NP}$, since we can just solve the problem for a no-certificate.

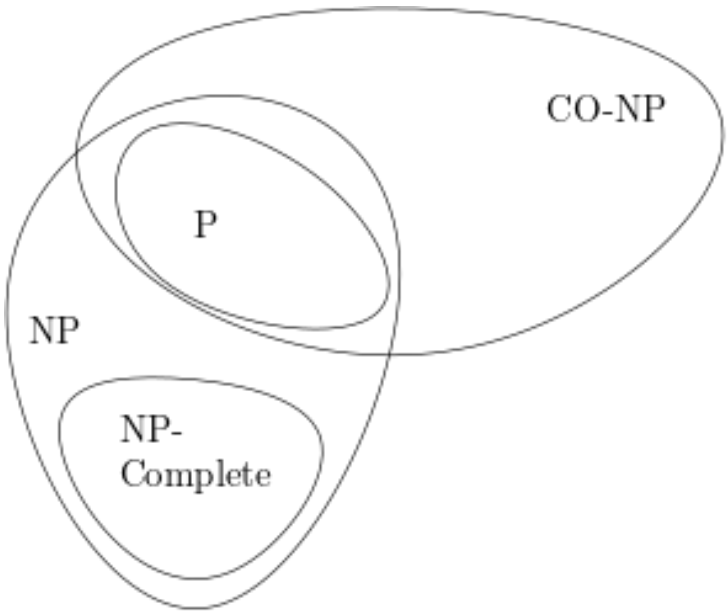


Figure 12: $\text{NP} \cap \text{CO-NP} = \text{P}$?

Definition 10.3. Problem that have both YES and NO certificates, i.e. problem in NP∩CO-NP are said to be **well characterized** or said to have a **good characterization**

Let us consider the decision version of LP: Given a feasible and bounded maximization LP, is there a feasible solution whose value is at least k?

We already know that there is a YES-certificate, as we can just check the feasible solution whose value is at least k, i.e. $\text{LP} \in \text{NP}$.

Remark 10.4 — Note that we did not have to know that there is a poly-time solution for LP, as we are only looking at verifying the YES-certificates

To see whether the LP problem lies in CO-NP, we can consider the dual of the LP. A NO-certificate would be any feasible solution of the dual whose value is less than k, meaning that LP $\in$ CO-NP.

This means that LP is a well-characterized problem. Once again, note that we did not have to use the fact that there is a poly-time algorithm for LP. Historically, when a problem is found to be well-characterized, it is likely that it belongs to P.

Decision Problem for Max Flow: Given a flow network, is there a feasible flow of value greater than or equal to k?

- Feasible flow is a yes-certificate, so this problem belongs to NP.
- The no-certificate is cut, as if there does not exist a flow greater than or equal to k, then there is a cut of value less than k.
- Note that once again we have shown that this problem is well-characterized, without using the knowledge of having a poly-time solution.

Decision Problem for Bipartite Perfect Matching: Given a bipartite graph G with bipartition (A, B) such that $|A| = |B|$, does G have a perfect matching?

- Yes-certificate is just the perfect matching which can be easily checked
- No-certificate: we can use Hall's Theorem, i.e. show that:

$$\exists X \subseteq A, \text{ s.t. } |N(X)| < |X|.$$

  We could also use Konig's Theorem:

$$\exists \text{ vertex cover of size } < n.$$

- As such this problem is well-characterized.

Problems that are well-characterized typically have an associated min-max relationship. These relationships are:

- beautiful and powerful cominatorial results
- lead to poly-time algorithms that are designed around it
- special cases of LP-duality.

We will now look at a well-characterized problem for which we do not know if there is a poly-time algorithm yet - factorization.

10.2 Well-characterization of Factorization

The decision problem for factorization is:

Given two integers x and y, does x have a factor less than y and greater than 1.

To find its first factor, find less than x and greater than 1. If no, then it is prime, if yes, then we can do binary search, but with $y = \frac{x}{2}$. After we find a factor, we can divide it and find the next factor.

Note that it takes $\log x$ calls to find a factor and since 2 is the smallest factor, there can be at most $\log x$ factors ($2 \times 2 \times \ldots \times 2$). As such this will take $\log^2 x$ calls. Since the input is of the order $\log x$, we would be able to check the yes-certificate.

The no-certificate is the prime factorization of x.

1. Check each prime factor and see if it is $\geq y$.
2. We must check if each factor is indeed prime. We can do this with a black box since there is a poly-time algorithm for checking whether a number is prime (Manidra Agarwal).
3. Multiply them together to see if their product is x.
4. After doing this, you should be convinced that the answer is no.

10.3 MAX-SAT Problem

SAT (Satisfiability) was the first problem to be shown was NP-Complete (Proved by Stephen Cook in the early 1970's)
Input:

- n boolean variables (each can be set True or False)
- m clauses: $C_1, C_2, \ldots, C_m$ (each is a disjunction of literals)

Goal: Find an assignment of true/false to the variables that maximizes the number of satisfied clauses.

Remark 10.5 — The original satisfiability problem: Given any boolean formula include "and", "or", "not", can the formula be "satisfied"?

Example 10.6
Let the boolean formula be:

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_4)$$

Can we assign x_i true or false to have the formula be true?

It turns out there this is equivalent (poly-time transformation) from this problem to MAX-SAT.

Definition 10.7. x_1 is a **variable**, and $\overline{x_1}$ is the **negation of a variable**. Collectively, they are **literals**

Definition 10.8. $\vee$: or, **disjunction**
$\wedge$: and, **conjunction**

Definition 10.9. A **clause** is one or more literals combined with disjunctions.

Definition 10.10. A **formula** is one or mor clauses combined with conjunctions

We can rewrite SAT as: Given a set of clauses, is there an assignment of true/false to variable, such that all the clauses are satisfied.

Remark 10.11 — This is clearly in the class NP, as if the answer is yes, we can just give the solution.

MAX-SAT is the maximization version of this problem (note that this is also NP-Hard, as we can easily use this to solve SAT).

10.4 Randomized Algorithm 1

Algorithm

Set each variable x_i to true independently with probability $\frac{1}{2}$.

Analysis

- Let $C_j = x_1 \vee x_2 \vee \ldots \vee x_k$.
- The probability that the clause is satisfied is $1 - \frac{1}{2^k} \geq \frac{1}{2}$.
- The expected number of clauses satisfied is:

$$\geq \frac{1}{2}m \geq \frac{1}{2}OPT.$$

Remark 10.12 — Consider the variable of the problem in which each clause has exactly 3 literals (MAX-3SAT). For this, we have:

$$\text{EXP } \# \text{ of clauses satisfied} \geq \frac{7}{8}m,$$

meaning that the approximation factor would be $\frac{7}{8}$.

Theorem 10.13
If there is a $\left(\frac{7}{8} + \epsilon\right)$- approximation algorithm for MAX-3SAT for any constant $\epsilon > 0$, then P=NP.

This means that our simple algorithm is the best possible for MAX-3SAT.

11 May 9th, 2019

11.1 MAX-2SAT

For each variable x_i, introduce a baraible y_i which is either +1 or -1. Also introduce an extra variable y_0, whihc is also either +1 or -1.

Intuition

Variable x_i is true of $y_i = y_0$ and x_1 is false if $y_i \neq y_0$. Consider the following clauses:

$$v(x_i) \to \frac{1}{2}(1 + y_i y_0).$$

As such, we can write the objective function as:

$$\max \frac{1}{2} \sum_i + \frac{1}{2}(1 + y_i y_0).$$

However, we need to consider the negation. Note that if it is we have a $\overline{x}_j$ clause, then we we would have:

$$v(\overline{x}_j) \to \frac{1}{2}(1 - y_i y_0).$$

As such, we have:

$$\max \sum v(\text{clause}_i).$$

Since the objective function contains the product, then we can change this into a vector program, since it can correspond to the dot product of two vectors. This is why we introduce y_0.

Checking Validity

$$v(x_i \vee x_j) = 1 - v(\overline{x}_i)v(\overline{x}_j) = 1 - \frac{1}{2}(1 - y_i y_0)\frac{1}{2}(1 - y_j y_0)$$

$$= 1 - \frac{1}{4}\left[1 - y_i y_0 - y_j y_0 + y_i y_j y_0^2\right].$$

Note that $y_0^2 = 1$:

$$\frac{3}{4} + \frac{1}{4} y_i y_0 + \frac{1}{4} y_j y_0 - \frac{1}{4} y_i y_j$$

$$= \frac{(1 + y_i y_0) + (1 + y_j y_0) + (1 - y_i y_j)}{4} = \frac{1}{4}(1.$$

Which is of the dot product. TODO: compute $v(\overline{x}_i \vee x_j)$ and other combinations As such, the objective function is of form:

$$\max \sum_{0 \leq i < j \leq n} \left[a_{ij}(1 + y_i y_j) + b_{ij}(1 - y_i y_j)\right],$$

with some coefficient a_{ij}, b_{ij}. Note that $i < j$ because of y_0

Constraints

Since $x_i \in \{0, 1\}$, we have

$$y_i^2 = +1, \text{ for } 0 \leq i \leq n.$$

Note that the MAX-2SAT is the special case of this problem.

11.2 Vector Program for MAX-2SAT

Relax to get vector program:

$$\max \sum_{0 \leq i < j \leq n} a_{ij}(1 + v_i \cdot v_j) + b_{ij}(1 - v_i \cdot v_j).$$

Subject to:

$$v_i \cdot v_i = 1$$
$$v_i \in \mathbb{R}^{n+1}.$$

Assume we have solved this exactly, and now we will round it. We will now round it. First lets make some modifications:

$$W = \max 2 \sum_{0 \leq i < j \leq n} a_{ij} \frac{1 + y_i y_j}{2} + b_{ij} \frac{1 - y_i y_j}{2}.$$

$$E(W) = 2 \sum a_{ij} \Pr(y_i = y_j) + b_{ij} \Pr(y_i \neq y_j).$$

This is the expected number of clauses satisfied by our algorithm. Comparing this to our vector program, we have:

$$\max 2 \sum_{0 \leq i < j \leq n} a_{ij} \frac{1 + v_i \cdot v_j}{2} + b_{ij} \frac{1 - v_i \cdot v_j}{2}.$$

Note that we want our randomized rounding to have the following properties:

- If two vectors v_i and v_j are pointing in the same direction, $\Pr(y_i = y_j)$ should be close to 1.
- If two vectors are in opposite direction, then $\Pr(y_i \neq y_j)$ is close to 1.

This is similar to the max-cut analysis, in which we take a random hyper plane to separate. Last time we analyzed that:

$$\Pr(y_i \neq y_j) = \frac{\theta}{\pi}.$$

11.3 Approximation Factor

For this, we have to show:

$$\frac{\Pr(y_i = y)j}{\frac{(1+v_i \cdot v_j)}{2}} \geq 0.878 \text{ and } \frac{\Pr(y_i \neq y)j}{\frac{(1v_i \cdot v_j)}{2}} \geq 0.878.$$

Which will give us an approximation factor of 0.878. As we know, we have:

$$\frac{1 - v_i \cdot v_j}{2} = \frac{1 - \cos(\theta_{ij})}{2} = \dots.$$

Next, we have:

$$\Pr(y_i = y_j) = 1 - \frac{\theta_{ij}}{2}.$$

Comparing with $\frac{1+\cos(\theta_{ij})}{2}$. What we know is that:

$$\frac{\frac{\theta}{\pi}}{\frac{1-\cos(\theta)}{2}} \geq 0.878.$$

Which is true for any θ. Replacing θ by $\pi - \theta$, we get the desired inequality.

Remark 11.1 — This is very similar to the max-cut, but the difference is we have to introduce a new variable y_0.

Remark 11.2 — When we used integer programming for the max-cut, we got nothing, but for MAX-SAT, we can get good results too, getting an approx ratio of 0.75.

11.4 MAX-2SAT But with Integer Linear Programming

For each boolean variable x_i, introduce a 0/1 variable y_i. For clause j, introduce 0/1 variable $z_j = 1$ if clause is true. If our clauses are:

$$(x_1 \vee \overline{x}_2 \vee x_3$$
$$\overline{x}_1 \vee x_2 \vee \overline{x}_3.$$

Our ILP would be is:

$$\max_j z_j$$

Subject to:

$$z_1 =\leq y_1 + (1 - y_2) + y_3 z_2 \leq (1 - y_1) + y_2 + (1 - y_3)$$
$$z_j \in \{0, 1\}$$
$$y_i \in \{0, 1\}.$$

Relaxing would give us would be is:

$$\max_j z_j$$

Subject to:

$$z_1 =\leq y_1 + (1 - y_2) + y_3 z_2 \leq (1 - y_1) + y_2 + (1 - y_3)$$
$$0 \leq z_j \leq 1$$
$$0 \leq y_i \leq 1.$$

Randomized Rounding: Each variable x_i is set to true independently with probability y_i (after solving relaxed LP). Otherwise set to false. Consider a clause with k literals, $x_1 \vee x_2 \vee \ldots v_k$ (no negation but analysis would hold still) We have:

$$\Pr(\text{Clause is satisfied}) = 1 - (1 - y_1)(1 - y_2)\ldots(1 - y_k).$$

We know that: $y_1 + y_2 + \ldots + y_j \geq z_j$. In the worse case, we have :

$$\Pr(\text{Clause is satisfied}) \geq 1 - \left(1 - \frac{z_j}{k}\right)^k \geq 1 - \left(1 - \frac{1}{k}\right)^k z_j.$$

For high k this approaches an approximation factor of $1 - \frac{1}{e}$. For MAX-2SAT, this is $\frac{3}{4}$.

Final Coverage

- steiner forest
- multiway cut
- semi-definite programming
- well-characterized problems

Index