Santucci's CME 305 Lecture Notes[1]

Lecture 1: Global Min Cut

Intro Graphs are simply a way to reason about things connecting other things.

$$G(V,E), \quad E \subseteq V \times V, \quad n = |V|, m = |E|$$

Cut A non-empty set of nodes (a subset) $S \subseteq V$.

Cut Size The sum of the weights from S to $V \backslash S$

Global Min-Cut A subset of nodes such that the cut has smallest size.

When do min-cuts appear? Computer networks, water networks, traffic networks.

Failed Approach What if we tried to find the min-cut through simulation of all possible cuts? To create/realize a cut S, we could flip a coin for each vertex for inclusion in S.

Realize that if we flip a coin for inclusion in S for *each vertex*, there are 2^n possible cuts we could realize. From this set we ignore the empty set and its complement, since these do not partition the set of vertices into two non-empty subsets.

Where does our approach fail? First, we define a

Clique: a graph with all edges present. Drawn as a hairball.

Dumb-Bell Graph: Two cliques connected by a single edge. Observe that there are only 2 ways with the dumb-bell graph that we can realize the global min cut. Hence

$$\Pr(S\text{is the global min cut}) = \frac{2}{2^n - 2}.$$

This is exponentially small. In expectation, we need to perform our simulation exponentially many times before we get a min-cut realization.

What if we use an edge-contraction?

A Better Idea

Algorithm 1: Karger's Contraction Algorithm

```
while # nodes > 2 do
    Pick edge e uniformly at random
    Contract e
end
```

Each contraction reduces the number of vertices by 1.

We often introduce a multi-graph in the process.

Degree The number of outgoing edges.

What's the probability of a single run realizing a global min-cut? Fix some global min-cut C. Assume that it has k edges crossing the cut.

$$\Pr(\text{failure on step 1}) = \frac{k}{|E|}$$

Handshake Lemma $\sum_{i=1}^{n} \deg(i) = 2|E|$.

We know that $k \leq \min_i \deg(i) \implies \frac{1}{2}\sum_{i=1}^{n} k \leq |E|$ hence $nk/2 \leq |E|$.

Hence $\Pr(\text{failure on step 1}) = \frac{k}{|E|} \leq \frac{k}{nk/2} = \frac{2}{n}$.

To realize a cut, we perform the contraction operation $n-2$ times. Let A denote the event that we succeed in realizing C, a min-cut, and let α_i denote the probability that we have "success" in contraction i (that is, we avoid contracting an edge which leaves our cut).

$$\begin{aligned} p = \Pr(A) &= \Pr(\alpha_1) \cdot \Pr(\alpha_2|\alpha_1) \cdot \ldots \cdot \Pr(\alpha_{n-2}|\alpha_{n-3}), \\ &\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right) \cdot \ldots \cdot \left(1 - \frac{2}{4}\right)\left(1 - \frac{2}{3}\right) \\ &= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\cdots\left(\frac{3}{5}\right)\left(\frac{2}{4}\right)\left(\frac{1}{3}\right) \\ &= \frac{2}{n(n-1)} = \binom{n}{2}^{-1} \end{aligned}$$

How many times do we have to repeat our algorithm? Suppose we repeat t times. The chance of failing to find C each time is

$$\Pr(\text{NOT finding } C) \leq (1-p)^t \leq e^{-tp},$$

where the last inequality simply follows from examining the function e^{-x} and $1-x$ on the interval $[0, x]$.

Set t large enough, e.g. $t = 100\binom{n}{2} = O(n^2)$, then

$\Pr(\text{NOT finding } C) \leq e^{-tp} = \frac{1}{e^{100}}$.

There are $n-2$ contractions per Karger. If we run t iterations of Karger, where each run of Karger can take $O(m)$ time (MST), we see that $O(t(n-2)m) = O(n^3 m)$ since $t = O(n^2)$.

Monte Carlo and Las Vegas This is an example of a Monte Carlo algorithm. With Monte Carlo - run time deterministic but probability of being correct is less than 1.

With Las Vegas algorithms, output always correct, but with some small probability run-time is long.

Lecture 2: Ford-Fulkerson

Assumptions All graphs today will be directed and weighted.

***s-t* min-cut** Suppose we want to disconnect the roadways between LA and NY. What would be the most efficient way to "bomb" them such that you can't drive from LA to NY.

Directed Cut Size The # of edges which *leave* the subset of vertices.

We can make any un-directed graph directed by replacing each edge with corresponding edges in each direction.

***s-t* flow** Respecting edge capacity, route items from s to t.

(i) Directions matter (can only move in edge direction)

(ii) Flow in = Flow out (except source and target nodes)

If all outgoing edges from the source are used/saturated, there's no way to ship more stuff from s.

To find max-flow via Ford-Fulkerson, we find a flow which saturates some directed cut (i.e. all edges which leave the cut are saturated), thus certifying that what it has found is a Max-Flow.

Theorem 1. *All s-t flows are less than (or equal to) all s-t directed cut sizes.*

The take-away is that every single s-t cut is an upper bound on s-t flow.

Residual Graph Given a graph and some flow, we define f_e as the flow on an edge, c_e as the capacity for an edge, and

Algorithm 2: Residual Graph

```
for each (u, v) ∈ E(G) do
    if f_e < c_e then
        Add (u, v) with capacity c_e − f_e        // forward edge
    end
    if f_e > 0 then
        Add (v, u) with capacity f_e              // backward edge
    end
end
```

Theorem 2. *s-t paths in residual graph correspond to legal flow route paths in G.*

Algorithm 3: Ford-Fulkerson

```
f_e ← 0 for all e ∈ E(G)
while ∃ s-t path in R_f(G) do
    Ship 1 unit of flow from s to t using path.
end
```

What's the idea behind the proof of correctness? Ford-Fulkerson finds a saturated cut, which is therefore an optimal max-flow.

Proof. When FF terminates, it creates some flow $f*$. Let Z be the subset of nodes *reachable* from s in $R_{f*}(G)$. We also consider the nodes in the complement of Z. Examine the edges in the original graph which correspond to this cut. Now, for all outgoing edges, they will be saturated in the $R_{f*}(G)$. For all incoming edges, they will have 0 flow from $Z \backslash V$ to Z.

From this, we know that $f*$ value is equal to the cut-value of Z. We now have an optimality certificate for $f*$. □

Note that since flows are bounded above by cuts, we now have an optimality certificate for Z, i.e. Z is a min-cut.

Run-Time Suppose c is the size of the s-t min-cut. To find an s-t path in a graph is $O(m)$ via DFS.

$O(mc)$ if we use any s-t path.

$O(nm^2)$ via shortest-paths and Edmunds-Karp.

$O(nm)$ via a 2012 result.

Deterministic Min-Cut Note that we could consider all $\binom{n}{2}$ s,t pairs and find the max-flow for each, then taking the minimum we have found our min-cut.

We can also realize there are $n-1$ distinct values the max-flow can take on (problem set #1).

Lecture 3: Matching, Edge-Disjoint Paths, Probabalistic Method

Bipartite Graph A graph whose vertices are split into two sets A, B such that all edges are between A and B and no edges are in A or B. Bipartite graphs often make problems easier. Graphs here are undirected, unweighted, and simple.

Matching A subset of edges $m \subseteq E$ is a matching $\iff$ no two edges in m are incident on the same vertex.

e.g. students and employers: each employer wants one employee, each employee can work at most one job; monogamous marriage problem.

Note that it's possible for an edge $(u,v) \notin G$; it's also possible that edge $(u,v) \in G$ but $(u,v) \notin H$, our subgraph.

Perfect Matching Each person gets married. Note that a requirement for this to happen is that $|A| = |B|$.

e.g. ICME Extend networking event.

Algorithm 4: Recipe: Max-Flow Min-Cut Probs.

1. Decide which of max or min to use.
2. Where to put source and target.
3. How to encode constraints into flows and cuts.
4. How to recover the solution.

Proposition 3. *If capacities are integers, then the max-flow is integer.*

Proof. Ford-Fulkerson saturates a cut. Each edge weight is an integer, hence the sum of edge weights are integers. □

Example: Maximal Matching Matching students to jobs. Each edge has capacity 1 (from s to students, between students and jobs, and between jobs and t). FF tells us which edges are saturated. That gives us a cut.

Algorithm 5: Maximal Matching via Max-Flow

1. Run s-t max flow.
2. Get a legal matching.
3. Have the largest # edges selected.

Example: max-weight perfect matching.

Example: Edge Disjoint Paths A set of paths in which no edge is repeated. Formally, a set of paths from a to b are *edge disjoint* $\iff$ they share no edges.

(Practical) Question: How many ways can we get from SF to LA if no roads can be reused? I.e. how many edge disjoint paths are there in a graph?

Answer: Set all capacities to 1. Find max-flow from a to b.

Proof. $\rightarrow$) ($\leq$) If there are k disjoint paths, then each of them may be used to route 1 unit of flow from s to t.

$\leftarrow$) ($\geq$) Suppose there is a k-flow. This may be generated via FF. Each time, we pump one unit of flow to yield an s-t path, which runs for k iterations. Note that FF paths on the residual graph are edge disjoint because of unit capacities; previous paths taken through R_G may be modified by future paths, but always in an edge-disjoint manner because of (a) the fact that each edge has unit capacity and (b) each iteration of FF strictly increases flow. □

[1] Based on Reza Zadeh's Winter 2016 course. Any typos are my own.

To run FF on an undirected graph, for each edge we simply create one forward directed edge (with capacity 1, if the graph was unweighted) and one "backward" directed edge, also with capacity 1.

Probabalistic Method Consider the simplest eample. If

$$\Pr(1 \text{ appears on a 6 sided die}) = \frac{1}{6} > 0,$$

then there is a face of the die with a one.

Example: Cut Size Every graph has a cut of the nodes such that at least $m/2$ edges appear in the cut.

Algorithm 6: Max-Cut (Random)

```
Input: A graph G(V, E) C ← {}.
for i = 1,2,..., n do
    with probability 1/2: C ← C ∪ {v_i}
end
```

Proof. Consider creating a cut C at random, by iterating over vertices and for each, include it in the cut C with probability 1/2. Notice that an edge is in a cut if and only if its incident vertices lie in separate partitions of the graph. There are exactly two ways this can happen, each with probability 1/4. Hence via this schema, each edge is included in our graph with probability 1/2.

Let $x = \sum_{i=1}^{m} \mathbf{1}\{\text{edge i is cut}\}$. Then,

$$E[x] = \sum_{i=1}^{m} E[\mathbf{1}\{\text{edge i is cut}\}] = \sum_{i=1}^{m} \Pr(\text{edge i is cut}) = \sum_{i=1}^{m} \frac{1}{2} = m/2.$$

Hence the expected value of a cut is $m/2$.

Formally, we have found that $E[\text{cut}] = \sum_{i=0}^{m} \Pr(\text{cut} = i) \cdot i = m/2$. Expectation is a weighted average of cut sizes. Hence we deduce that our algorithm returns a cut of size at least $m/2$ with strictly positive probability. If not, it would contradict the equality between $E[\cdot]$ and our weighted average.

So, all probabilities for cut-values greater than $m/2$ cannot all be 0 simultaneously, because if they were it would contradict weighted average equality.

Hence there exists a non-zero (strictly positive) probability which emits a cut with value *at least* $m/2$.

Hence there exists a cut out there with size $\geq m/2$. (There also exists a cut with size $\leq m/2$ using min-degree) □

Ramsey Number Our class has at least 49 people. In our class, there exists a group of 5 people such that either they all know each other (fully connected) *or* they are all strangers (fully disconnected).

Proposition 4. *Given $r > 0$. For sufficiently large n,* all *graphs on n nodes contain either K_r* or *$\bar{K}_r$ (a clique on r nodes, or a complement of a clique on r nodes).*

$R(r)$ is the smallest n for which the above proposition is true. $R(3) = 6$, $R(4) = 18$, $R(5) \in (43, 49)$, $R(6) \in (102, 165)$. In general,

$$\frac{c\sqrt{2}}{e} r 2^{r/2} \leq R(r) \leq r^{-\log r/\log\log r} \cdot 2^{2r}.$$

Review Global Min Cut has 1 value by definition. There are at most $\binom{n}{2}$ possible cuts which realize this value (since Karger spits out a cut, min-cut returned with probability $\binom{n}{2}^{-1}$. If there were more than $\binom{n}{2}$ min-cuts, we would violate the union bound.

s-t Min Cut has $n-1$ different values over choices of s, t. There are an exponential number of them for fixed s, t in general (see homework #2).

What about Max-Cuts? That's a hard-problem.

Lecture 4: Ramsey Numbers, Trees and MSTs

Ramsey Numbers For sufficiently large graphs on n nodes, there exists an induced subgraph of K_r or $\bar{K}_r$ (an independent set of size r).

$$\underbrace{2^{r/2}}_{\text{via Prob Method}} \leq R(r) \leq 2^{2r}.$$

(Upper bound is proved directly in Diestel) All graphs of size 2^{2^r} have this property. Graphs of size $2^{r/2}$ may not have this property.

Induced Subgraph An induced subgraph can be constructed by deleting vertices (and with them all incident edges), but *no more* edges. If additional edges are deleted, the subgraph is *not induced*

Erdos Renyi Random Graphs $G(n, p)$: for n positive integer and $p \in (0, 1)$ is a probability. Each edge exists i.i.d. with probability p.

If $p = \frac{\ln n}{n}$, the graph is almost surely connected. This is a threshold.

Notice that

$$\Pr(G(n,p) \text{ has a clique of size r}) \leq \binom{n}{r} p^{\binom{r}{2}},$$

$$\Pr(G(n,p) \text{ has an independent set of size r}) \leq \binom{n}{r} (1-p)^{\binom{r}{2}}.$$

Note that there are at most $\binom{n}{r}$ possible cliques; for each, we require all $\binom{r}{2}$ edges be added with probability p.

They *do* overlap (i.e. not mutually exclusive), hence we apply Union Bound:

So, if $p = 1/2$, then

$$\Pr(G(n,1/2) \text{has either } K_r \text{ or } \bar{K}_r) \leq 2\binom{n}{r}\left(\frac{1}{2}\right)^{\binom{r}{2}}.$$

If this probability is strictly less than one, then there exists graphs which satisfy neither. The assertion is *trivial* when $r = 1$, hence assume $r \geq 2$,

$$\Pr(A) \leq 2\binom{n}{r}\left(\frac{1}{2}\right)^{\binom{r}{2}} < 2\left(\frac{n^r}{2^r}\right)\left(\frac{1}{2}\right)^{\binom{r}{2}}.$$

The strict inequality follows from the fact that $r \geq 2$ and that

$$\binom{n}{r} = \frac{n!}{r!(n-r)!} = \frac{\overbrace{n(n-1)(n-2)\dots(n-r+1)}^{r \text{ terms}}}{\underbrace{r(r-1)(r-2)\dots(2)(1)}_{r \text{ terms}}} < \frac{n^r}{2^r}$$

Let $n = 2^{r/2}$. Then,

$$\Pr(A) \leq 2 \cdot \frac{2^{r^2/2}}{2^r} \cdot 2^{-r(r-1)/2} = 2 \cdot 2^{-r/2} < 1.$$

Since $\Pr(A) < 1$, by the probabalistic method there exists a subgraph of size $2^{r/2}$ which has neither K_r nor $\bar{K}_r$.

Cycle A path along edges such that the same vertex is visited twice.

A graph with no cycles is called a forest.

A connected forest is a tree.

A leaf is a node with degree 1.

Proposition 5. *Every longest path in a forest must have vertices with degree 1.*

Proof. Look at one end. Assume toward contradiction it does not have degree 1. If the node connects to any other node *along the path*, we have a contradiction: no cycles. If the node connects to another node off the path, then we have a contradiction: definition of longest path. □

Proposition 6. *There are at least 2-leaves in a tree ($n \geq 2$).*

Proof. Start at any node. Pick an edge we haven't seen before. At some point, we must hit a node we can't get past. This is a leaf of degree 1. Repeat this process starting at this leaf to find the other leaf. □

Trees

Theorem 7. *Any two of the three following statements implies the other. This defines a Tree.*

1. *G connected.*
2. *G has no cycles.*
3. *G has $n-1$ edges.*
4. *For $u, v \in V(G)$, G has exactly one u, v-path.*

Proof. 1: Connected, 2: acyclic, 3: $n-1$ edges, 4: unique path.

(1,2 $\implies$ 3) We use induction on n. Base case: $n = 1$. The acyclic 1-vertex graph has no edges. For $n > 1$, suppose the result true for graphs with fewer than $n-1$ vertices. Given any acyclic connected graph G, we can find a leaf v where $d(v) = 1$. Notice that $G' = G - v$ also acyclic and connected. By induction hypothesis, $e(G') = n-2$. Since only one edge is incident to v, we have $e(G) = n-1$.

(1,3 $\implies$ 2) We are given a graph G which is connected with $n-1$ edges. Suppose G contains at least a cycle. Delete edges from cycles one at a time to yield G', which is acyclic. A cut-edge is an edge whose deletion increases the number of components. No edge of a cycle is a cut-edge. Hence G' connected. Now, preceding paragraph implies $e(G') = n-1$, since it's connected and acyclic. But we are given $e(G) = n-1$ by (3). Hence no edges were deleted. Hence $G' = G$ and G is acyclic.

(2,3 $\implies$ 1) Let $G_1, \dots, G_k$ be the components of G. Since every vertex appears in one component, $\sum_i n(G_i) = n$. Since G has no cycles, each component satisfies the property that (each component is connected *and* each component has no cycles). Thus $e(G_i) = n(G_i) - 1$, and $e(G) = \sum_i e(G_i) = \sum_i [n(G_i) - 1] = n - k$. But we are given $e(G) = n-1$. So $k = 1$. Hence G connected.

(1,2 $\implies$ 4) Since G connected, each pair of vertices is connected by a path. Suppose a pair connected by more than one. We choose the shortest (total length) pair P, Q of distinct paths with the same endpoints. By this extremal choice, no internal vertex of P or Q can belong to the other path. Hence $P \cup Q$ is a cycle, which contradicts (1,2).

(4 $\implies$ 1,2) If there is exactly one u, v-path for every $u, v \in V(G)$, then G connected. Assume toward contradiction G has a cycle, C. Then G has two u, v-paths for $u, v \in V(C)$, which is a contradiction. Hence G acyclic. □

Spanning Trees A spanning tree of G is a subgraph of G that is a tree and has all nodes. The weight of a tree is the sum of the edges in the tree.

The Minimum Spanning Tree problem is to find the spanning tree of G with minimum weight. "The backbone of the graph".

Theorem 8 (Cut Property)**.** *The smallest edge leaving* any cut *must be in* all *Minimum Spanning Trees (assuming all edges have unique/distinct weights).*

Proof. Take any cut S (of the 2^n possible), and any minimal spanning tree, T. Since T connected, it *must* contain one of the edges of this cut.

Assume toward contradiction that an edge e is lowest weight but $e \notin T$. Consider the u, v-path from $u \in S$ to $v \in V \backslash S$. At some point, we must use an edge, call it t in order to "get across" the cut. Suppose we remove t in favor of e. Call this tree $T' = T \cup \{e\} \backslash \{t\}$. T' is another spanning tree. But then the weight of T' is less than the weight of T, contradicting that T is a minimal spanning tree. □

Kruskal's Algorithm Order the weights from smallest to largest, $e_1, \dots, e_m$. While not all n nodes are in the tree, if adding the edge e_j does not introduce a cycle, add it.

Algorithm 7: Kruskal's Algorithm

```
1 Order weights in increasing order.
2 T ← {}.
3 for e ∈ e_1 ≤ e_2 ≤ ... ≤ e_m do
4     if T ∪ {e} does not create a cycle then
5         T ← T ∪ {e}.
6     end
7 end
8 Return T.
```

We run Kruskal, and get a result T. Each edge we add, we invoke our theorem to get a certificate saying it must be the case that ...

Consider when we added edge $(u, v) = e \in E$. Consider state F, when we're in a forest, of Kruskal where e was added.

Let set S denote the set of nodes reachable from $u \in F$. Note that $v \notin S$ since if it were we would be introducing a cycle. Note that S forms a tree by construction.

Adding any (or all) of these edges does *not* create a cycle. Since we are considering them for the first time, they have the lowest weight. Hence e is the first edge from S that is being considered for inclusion. Hence e is the lowest weight candidate edge. Hence $e \in T$.

Removing Unique Weights Assumption To remove the assumption that edge weights must be unique, we may perturb each weight by a small amount. This leaves the MST value almost unchanged, but effectively breaks ties.

Lecture 5: Cycles, Circuits, Commute Times

Hamiltonian Cycle A simple cycle (i.e. no node is repeated, starts and ends at same node) which visits all nodes exactly once. NP Complete problem. Not all graphs have Hamiltonian cycle; e.g. trees.

Theorem 9. *If all degrees of the graph at least $n/2$, there exists a Hamiltonian Cycle.*

For a proof, see problem set #1. The idea is to start with a longest path of length k, show the neighbors of endpoints must lie in path, and that there then exists a path which traverses all nodes.

One consequence of this is that there exists a Perfect Matching. To see this, we simply walk along the Hamiltonian Cycle and alternate labeling vertices "red" and "blue". We ignore the last edge. Hence we get a perfect matching.

Eularian Circuit A cycle which visits all *edges* exactly once (not simple, nodes can be visited more than once). In polynomial time, we can determine whether a graph has an Eularian Circuit and in addition find one in particular.

Theorem 10. *A connected graph has a Eularian Circuit $\iff$ all degrees even.*

Proof. $\rightarrow$) Suppose there exists a node with odd degree. An Eularian Circuit requires that we use each node exactly once. But we can't do this without re-using edges, since each time we visit a node we also must be able to leave it. Hence we require the neighborhood size of each node to be even.

$\leftarrow$) Suppose we have all even degrees. We greedily start using edges from the graph. We may realize a cycle. The remaining nodes not yet visited all have even degree. Since G connected, there exists an untaken edge which connects the first cycle to the rest of the graph which has not yet been visited. Start with this edge, generate another cycle. We repeat, always looking for edges in an ever expanding Eularian Circuit which lead to untraversed edges. We continue to expand until all nodes have been visited / all cycles stitched together.

We know there exists a cycle since it's connected and for each node you visit, it has another unused outgoing edge. □

Random Walk We use the uniform random walk (a Drunken Traveler): at each node, choose an outgoing edge uniformly at random.

Hitting Time from v_i to v_j: the time it takes to arrive at node v_j for the first time, starting from node v_i.

$$h_{ij} = E[\text{time it takes RW to traverse from i to j}] \neq h_{ji}.$$

Commute Time $c_{ij} = h_{ij} + h_{ji} = c_{ji}$.

Cover Time of G from node u,

$$C_u(G) = E[\#\text{ steps to visit all nodes in G starting from u}].$$

We define the cover time for a graph to be

$$C(G) = \max_u C_u(G).$$

Cover Time of Complete Graph At first step, we see a new node with probability 1. At second step, there are $n-2$ nodes yet visited. We realize we have a sum of geometric Random Variables (how many times must we flip before getting heads)

$$A = \text{Geom}\left(\frac{n-1}{n-1}\right) + \text{Geom}\left(\frac{n-2}{n-1}\right) + \text{Geom}\left(\frac{n-3}{n-1}\right) + \ldots + \text{Geom}\left(\frac{1}{n-1}\right).$$

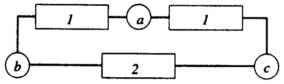
Note that we use $(n-1)$ in the denominator because there are no self-loops in the graph G, hence there are only $n-1$ possible options at each drunken step. Then,

$$E[A] = \sum_{i=1}^{n-1} E\left[\text{Geom}\left(\frac{n-i}{n-1}\right)\right] = \sum_{i=1}^{n-1} \frac{n-1}{n-i} = (n-1)\sum_{i=1}^{n-1} \frac{1}{n-i}$$

$$= (n-1)\sum_{i=1}^{n-1} \frac{1}{i} = (n-1)H_{n-1} = (n-1)\Theta(\log n) = \Theta(n \log n).$$

Effective Resistances

Effective Resistances What about general graphs. Can we get decent bounds on them? Yes! a *resistive electrical network* is an undirected graph; each edge has associated with it a positive real *branch resistance* (for us, this value is always 1, since we use 1 ohm resistors?).



If a current of one amp were injected into node b and removed from node c, using Kirchoff's Law and Ohm's Law yields the following: half an amp flows along branch bc, and the other half through branch ba and onto ac. The voltage difference between c and b is one volt, while the *voltage difference* between c and a (and between a and b) is half a volt.

We look at n, m and the effective resistance of the graph. (Formally, the *effective resistance* between two nodes u and v is the *voltage difference* between u and v when one ampere is injected into u and removed from v.) In our figure, the effective resistance between b and c is 1, whereas the branch resistance is 2.

Calculating Effective Resistance between two nodes

Series Rule: Consider a *series circuit*, e.g. a path of 5 resistors, the resistance is 5. In general, the Series Rule suggests the resistance between two nodes is the path length between them.

Parallel Circuits: We can also have a *parallel circuit* which has multiple paths from a to b. Let P_{ab} denote the set of paths from node a to node b. Then,

$$R_{ab} = \frac{1}{\sum_{i=1}^{\#\text{ ab paths}} 1/\text{length of path}}.$$

Shorting and Cutting

When we Short two nodes together, electrical resistance can't increase. Hence we realize a lower bound on effective resistance this way.

When we Cut an edge, i.e. take a 1-ohm resistor and replace it with an infinite capacity resistor, electrical resistances can't decrease. Hence we realize a upper bound.

(If we add an edge we realize a lower bound.)

Bounding Effective Resistances If there is an (a, b) edge then $R_{ab} \leq 1$.

Notice that R_{ab} can be large if a, b not directly connected.

The effective resistance between a and b is *at most* the length of the shortest path between them in G (bound is tight; chain graph).

Theorem 11. *For any two vertices u and v in G, the commute time $C_{uv} = h_{ij} + h_{ji} = 2mR_{uv}$.*

Proof. We are given an undirected simple graph (unweighted). We turn each node into a 1 ohm resistor.

For a vertex $x \in G$, let $\Gamma(x)$ denote the set of vertices in V adjacent to x, and let $d(x)$ denote its degree, $|\Gamma(x)|$.

We create **Circuit 1**. Let ϕ_{uv} denote the voltage at u in $\mathcal{N}(G)$ with respect to v, if $d(x)$ amps of current are injected into each node $x \in V$, and $2m$ amps are removed from node v (fixed node v). We first prove that for all $u \in V$,

$$h_{uv} = \phi_{uv}.$$

Using Kirchoff's Law (current in = current out) and Ohm's Law ($v = ir$), we obtain that for all $u \in V \backslash \{v\}$, and the fact that each edge has 1 ohm resistance (so $r = 1$ disappears as a coefficient),

$$\underbrace{d(u)}_{\text{current in}} = \underbrace{\sum_{w \in \Gamma(u)} (\phi_{uv} - \phi_{wv})}_{\text{current out}} \tag{1}$$

$$= d(u)\phi_{uv} - \sum_{w \in \Gamma(u)} \phi_{wv}$$

Hence we may rearrange to see that

$$\phi_{uv} = \sum_{w \in \Gamma(u)} \frac{1}{d(u)} \left[1 + \phi_{wv}\right]$$

where we note that $\phi_{vv} = 0$ using Kirchoff's Law.

By definition of expectation, for all $u \in V \backslash \{v\}$,

$$h_{uv} = \sum_{w \in \Gamma(u)} \frac{1}{d(u)} \left(1 + h_{wv}\right), \tag{2}$$

where we note that $h_{vv} = 0$.

Hence we have that $h_{uv} = \phi_{uv}$.

Circuit 2. Realize that h_{vu} is the voltage ϕ_{vu} at v in $\mathcal{N}(G)$ measured with respect to u, when currents are injected into all nodes and removed from u. To see this, first extract $d(x)$ amps from all nodes, then inject $2m$ amps at node u. Then we see that

$$\phi_{uv}^{\text{circuit 2}} = h_{vu}.$$

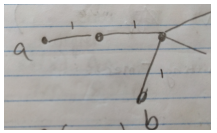
Changing signs, ϕ_{vu} is now the voltage at u relative to v when current is injected at u, and removed from all other nodes.

Since resistive networs are *linear*, we can determine C_{uv} by *super-imposing* (being careful with the sign) the networks on which ϕ_{uv} and ϕ_{vu} are measured.

Let C_1 and C_2 denote our circuit 1 and circuit 2 respectively. Then,

$$\phi_{uv}^{C_1+C_2} = h_{uv} + h_{vu} = c_{uv}.$$

Hence C_{uv} being the voltage between u and v when $\sum_{w \in V} d(w) = 2m$ amps are injected into u and removed from v, which yields the theorem by Ohm's Law since currents at all nodes except u and v cancel. □



Notice that $C_{ab} = 2(n-1) \cdot 3$ since the tree has $n-1$ edges and $R_{ab} = 3$.

Proposition 12. *In any n vertex graph, and for all vertices u, v,*

$$C_{uv} < n^3.$$

Proof. Notice that R_{uv} is at most the length of the shortest path between them in G. There are only n nodes in the graph. Hence the longest path starting at u ending at v can only visit $n-2$ other nodes before arriving at v (otherwise we visit a node more than once, and it's no longer a shortest path). Hence the shortest uv path is at most $n-1$. Hence in any n-vertex graph, $C_{uv} = 2mR_{uv} \leq 2m(n-1) = O(n^3)$. □

So regardless of where we start, commute time is bounded above by $2m(n-1) = O(n^3)$. For a tree, the commute time is $O(n^2)$ since $m = n-1$.

Notice that $O(n^3)$ is a tight bound. Consider a lollipop graph as an example. We have a clique of size $n/2$ and a chain of size $n/2$. We prove this cover time in the next lecture.

Cover Times We consider more scenarios.

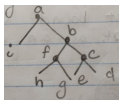
Corollary 13. *If two nodes u, v that are directly connected, then*

$$C_{uv} \leq 2m.$$

Proof. Since they are directly connected, there is at least one edge from u to v with resistance 1. Hence $R_{uv} \leq 1$. Hence $C_{uv} \leq 2m$. □

Can we use commute times to upper bound cover times? Notice that R_{ij} can be large if i, j not directly connected.

Theorem 14. *In any n-vertex graph, $C(G) \leq 2m(n-1)$.*



Proof. Start with any connected graph G. It has a spanning tree, by definition of being connected. By previous theorem, for any two *adjacent nodes*, u, v, $C_{uv} \leq 2m$. Notice that if we use post-order traversal to visit each node in our tree, the expected time it takes to do this is *at least* as long as the expected time it takes to visit each node in the tree (not necessarily in order). Hence

$$\text{Cover Time}(u) \leq h_{ab}+h_{bc}+h_{cd}+h_{dc}+h_{ce}+h_{ec}+\ldots+h_{fb}+h_{ba}+h_{ai}+h_{ia}.$$

Hence,

$$\text{Cover Time}(u) \leq \sum_{i,j} h_{ij}.$$

Notice that $h_{cd} + h_{dc} = C_{cd}$. So replace all "pairs" with commute times. Hence

$$\begin{aligned}\text{Cover Time}(u) &\leq C_{ab} + C_{bc} + C_{cd} + \ldots + C_{ai} \\ &\leq 2m + 2m + 2m + \ldots + 2m \\ &= (n-1)2m\end{aligned}$$

□

Lecture 6: Cover Times and Effective Resistances

Background Recall the following definitions.

Asymptotic Upper Bound: we say that $T(n)$ is $O(f(n))$ if there exists constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, we have $T(n) \leq c \cdot f(n)$.

Asymptotic Lower Bound: we say that $T(n)$ is $\Omega(f(n))$ if there exist constants $\epsilon > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, we have $T(n) \geq \epsilon \cdot f(n)$. Note that ϵ is a fixed constant, independent of n.

Asymptotically Tight Bound: If a function $T(n)$ is both $O(f(n))$ and $\Omega(f(n))$, we say that $T(n)$ is $\Theta(f(n))$

And in total abuse of notation, we see that $o(\cdot)$ is asymptotic $<$, and $w(\cdot)$ is asymptotically $>$.

What if we use our bound $C(G) \leq 2m(n-1)$ for a complete graph. We get a bound of $O(n^3)$, but we showed earlier that in fact the cover time is $\Theta(n \log n)$. This is a bad bound.

For trees, we use $C(G) \leq 2m(n-1) = 2(n-1)(n-1) = O(n^2)$, which is correct.

Are there better upper/lower bounds?

Effective Resistance of a Graph Define, *effective resistance* for an entire graph by

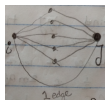
$$R(G) = \max_{u,v} R_{uv}.$$

It can be shown that

$$mR(G) \leq C(G) \leq e^3 2m \ln(n) R(G) + n$$

is within a $\log n$ factor of the exact cover time.

Show the bound is tight For example, consider a complete graph on n vertices, K_n. Consider a subset of edges as pictured below.



We have n vertices total, so $n-2$ vertices in the middle layer. There are $n-2$ paths each with resistance 2. Hence, when we consider these paths in addition to the direct (i,j) edge, we see that our effective resistance is bounded above by

$$R_{ij} \leq \frac{1}{1 + (n-2)(\frac{1}{2})} = \frac{2}{n} = O\left(\frac{1}{n}\right).$$

To be clear, by examining only a subset of edges of the complete graph, we are leaving out additional terms from the denominator which are positive. Hence the first inequality.

Plugging this into our upper bound at the start of the section, we see that we get a $O(n \log n)$ cover time, which is tight.

Cover Time not monotone w.r.t. edges Notice that there is no clear relation between cover time and $|E|$.

Start with a chain on $n/2$ vertices. We know that $C(G) \leq 2m(n/2-1) = O(n^2)$. We also know that $mR(G) \leq C(G)$. For a chain, $R(G) = (n-1)$. Hence $mR(G) = (n-1)(n-1)$, and we see that the cover time of a chain is $\Theta(n^2)$.

Now consider the lollipop graph with a clique of size $n/2$ joined to chain of size $n/2$. Notice that $C(G) \leq 2m(n-1) = O(n^3)$. Now consider $mR(G) \leq C(G)$. Notice that because R_{ab} upper bounded by the length of the shortest path between them, and the longest shortest path in lollipop is $n/2$, then we see that $mR(G) = O(n^2) \cdot O(n) = O(n^3) \leq C(G)$. Hence $C(G) = \Omega(n^3)$, and $C(G) = \Theta(n^3)$.

If you know max flow between u and v, you can compute effective resistances. Using these you get a bound on cover times.

Theorem 15. *Cover time is bounded below by number of edges times effective resistance of a graph.*

$$mR(G) \leq C(G).$$

Proof. Find i, j which maximize R_{ij}. We know that $C_{ij} = h_{ij} + h_{ji} = 2mR_{ij} = 2mR(G)$. Hence at least one of h_{ij} or h_{ji} is at least $1/2$ of $2mR(G)$ (if not, the sum is less than $2mR(G)$, and we get a contradiction). Hence (without loss of generality) we suppose $h_{ij} \geq mR(G)$ for some i, j, which we now fix. But clearly $C(G) \geq h_{ij}$ for any i, j. Hence $C(G) \geq h_{ij} \geq mR(G)$. □

Markov Inequality To prove the upper bound, we need more machinery. Turn to Markov's Inequality. For all random variables X which are *non-negative*,

$$\Pr(X \geq a) \leq \frac{E[X]}{a}.$$

Useful when $E[X]$ small.

Proof. Define a new random variable $\mathbf{1}\{x \geq a\} \cdot a$ (trivially, $\mathbf{1}\{x \geq a\} \cdot a \leq X$). Note that since it's an indicator, $E[\text{indicator variable}] = \Pr(\text{event})$. Hence,

$$E[\mathbf{1}\{x \geq a\} \cdot a] \leq E[X],$$

which implies that $\Pr(X \geq a) \leq E[X]/a$. □

Example Suppose X_{uv} is the number of steps it takes to go from node u to node v. Clearly, $E[X_{uv}] = h_{uv}$. Then,

$$\Pr(X \geq 10h_{uv}) \leq \frac{h_{uv}}{10h_{uv}} = \frac{1}{10}.$$

Theorem 16.

$$C(G) \leq e^3 m \ln(n) R(G) + n.$$

Proof. Consider a random walk which is split into $\ln n = \log_e n$ blocks, each of size $2e^3 mR(G)$. Each block is an epoch.

Fix some node v. What's the probability that we don't visit v in an Epoch? Each epoch starts at some node u. Note that *regardless of* u, the time it takes to get from u to v is bounded $h_{uv} \leq 2mR(G) = C_{uv} = h_{uv} + h_{vu}$.

Let X_{uv} denote the number of steps from u to v.

$$\begin{aligned}&\Pr(\text{v not visited in a particular epoch}) \\ &= \Pr(X_{uv} \geq 2me^3 R(G)) \\ &\leq \Pr(X_{uv} \geq e^3 h_{uv}) && \text{since } h_{uv} \leq 2mR(G). \\ &\leq \frac{h_{uv}}{e^3 h_{uv}} && \text{by Markov} \\ &= \frac{1}{e^3}\end{aligned}$$

Now, since we have $\log_e(n)$ epochs, for fixed v,

$$\Pr(\text{v not visited in } \textit{any} \text{ epoch}) \leq \left(\frac{1}{e^3}\right)^{\ln n} = \frac{1}{n^3}.$$

Note that this follows from the fact that if we don't know where we started in an epoch, knowing whether we visited v in the current epoch does not tell us about where we will hit v in any other epoch. Hence we have independence of the events.

Now, we use the Union Bound to pin down $\Pr(v_1 \text{ not visited} \cup \ldots \cup v_n \text{ not visited})$,

$$\Pr(\exists v \text{ not visited in any epoch}) \leq \frac{n}{n^3} = \frac{1}{n^2}.$$

So, run a walk of length $2e^3 mR(G) \log_e n$.

If we haven't visited all nodes yet, run for further n^3 time.

$$E[\text{total cover time}] \leq 2e^3 mR(G) \ln n + \underbrace{\frac{1}{n^2} \cdot n^3}_{=n}.$$

□

Boolean Formula Satisfiability We have a set of variables $X_1, X_2, \ldots, X_n$ which take values from the set {true, false}. Recall the $\wedge, \vee, \neg$ denote the `and`, `or`, and `not` operators respectively.

SAT: Given a formula on v variables, can $X_1, X_2, \ldots, X_n$ be assigned such that the formula evaluates `true`.

In general, no poly-time solution. It's NP-Hard. For $k = 1$ or $k = 2$ we may solve the problem in polynomial time using a *Monte Carlo* algorithm.

`AND` – conjunction, `OR` – disjunction, `NOT` – negation. A formula is in *conjunctive normal form* if it is a conjunction of clauses (or a single clause).

Using laws of Boolean Algebra, we may transform every propositional logic formula into an equivalent conjunctive normal form. However, it may be exponentially longer.

2-SAT, Monte-Carlo Consider the set of formulas that are a conjunction of unions with each clause having exactly 2-literals.

e.g. $(X_1 \vee X_3) \wedge (X_3 \vee \neg X_2) \wedge (\not{X}_{37} \vee \not{X}_1)$. X_1 and $\neg X_1$ are called *literals*.

Algorithm 8: 2-Sat

```
Initialize B = {X_1, X_2, ..., X_n} randomly.
while there exists an unsatisfied clause do
    Fix attention to an arbitrary unsatisfied clause.
    Pick one of two literals uniformly at random.
    Complement the value of the chosen literal.
end
```

How long does it take for us to find a satisfying assignment if one exists?

Assume that formula is satisfiable with an assignment A. Refer to the assigned by A to be the "correct values". Let n be the number of variables in an instance. The progress of this algorithm can be represented by particle moving along the integers $\{0, 1, 2, \ldots, n\}$. The position indicates how many variables in the current solution have the correct values. At each iteration, we complement the current value of one of the literals of some unsatisfied clause.

Hence the particles position must change by 1 at each step. A particle at position i, for $0 < i < n$, can only move to $i-1$ or $i+1$. A particle at 0 can only move to 1. The process terminates when the particle reaches position n. It may also terminate at some other position with a satisfying assignment other than A.

Observation In an unsatisfied clause, at least one of the two literals has an incorrect value relative to our optimal solution A. Hence with probability at least $1/2$, we increase (by one) the number of variables having their correct values. The motion of the particle is thus a random walk on the line.

Notice that the cover time of this graph is $O(n^2)$, since $C(G) \leq 2m(n-1) = 2(n-1)(n-1) = O(n^2)$. Hence the expected time it takes to traverse from 0 to n is n^2 steps. If you find yourself running the above algorithm $O(n^3)$ times without finding a solution, the formula is probably not satisfiable.

For example, $\Pr(X \geq 3n^2) \leq \frac{1}{3n}$ and so with very small probability we may be wrong if we go ahead and exit the algorithm and conclude the problem is not satisfiable.

Maximum Satisfiability From Ch 5, MR. Suppose we simply want to maximize the number of satisfied clauses (rather than decide whether there exists an assignment which satisfies the problem).

Theorem 17. *For any set of m clauses, there is a truth assignment for the variables that satisfies at least $m/2$ of the clauses.*

Proof. Each variable is set to `true` or `false` independently and with probability 1/2. For $1 \leq i \leq m$, let $Z_i = 1$ if the ith clause is satisfied and 0 otherwise. For any clause containing k literals, the probability that it is *not* satisfied by this random assignment is 2^{-k}, since this event takes place if and only if each literal gets a specific value, and the (distinct) literals in a clause are assigned independent values. This implies the probability that a clause with k literals is satisfied is $1 - 2^{-k} \geq 1/2$. Hence $E[Z_i] \geq 1/2$ for all i. Hence in expectation, we satisfy $\sum_{i=1}^{m} E[Z_i] \geq m/2$ clauses by this random assignment. Thus, there exists at least one assignment of values to the variables for which $\sum_{i=1}^{m} Z_i \geq m/2$, by a fundamental theorem of the probabalistic method. □

Lecture 7: NP Hard Problems, Reductions

Problems are NOT Algorithms There can be many algorithms which solve one problem.

Problem: An (infinite) listing of input → answer (of the problem). i.e. a problem is defined by its answers.

For example the global min cut problem is an infinite listing of answers, where for each graph we have an answer.

Algorithm: A finite sequence of operations on a computer.

Solve: An algorithm *solves* a problem if the algorithm can find the problem answer to any input.

Decision Problem: A problem with only "yes" or "no" answers.

Almost everything can be written as a decision problem. e.g. does this graph have a min-cut of size k? Does it have a Hamiltonian cycle?

Verify: Verifying answer a to an input x means checking that an answer to problem input x is a (for us, here a is a "yes" or "no" problem). This is *different* from solving the problem. We simply verify for fixed x and a. Whereas the algorithm must solve for all x and a.

A *verifier* is a program to verify an answer.

Verifier (Algorithm) for problem x. Takes as input x and a candidate answer a (a yes or a no). Checks if the problem answer to x is indeed a.

A verifier can take some time; i.e. it has run-time. It performs some number of steps and determines if you are right.

Certified Verifier A verifier which in addition to x and a also takes a certificate c to verify if a is the answer to x.

E.g. Hamiltonian Cycle in the graph? The certificate could be the cycle itself (if one exists).

Notice this is *asymmetric*. We can only have a certified verifier for decision problems which have *yes* answers.

A certificate is just bits, 0's and 1's. It is *not* an algorithm.

Verifiers are algorithms. Certificates just take space. We'd like for these to be polynomial.

NP Problems: Decision problems which have certified verifiers with poly run-time (the verifier runs itself runs in poly-time), and polynomial certificate size.

That is, we only require that we can "check" your answer in polynomial time.

> A decision problem is in NP $\iff$ its "yes" answer can be verified in poly-time with poly-sized certificate.

Max-Cut Problem Input, $G(V, E)$ and a threshold t. Decision: Is there a cut with size at least t in G? Certified Verifier: *certificate* - a cut $S \subseteq V$, and a *verifier algorithm* - computes `cut`(S) and determines how it compares with t.

Any optimization problem whose objective and constraints can be evaluated in poly-time is in NP. This also requires poly-number of variables.

Any optimization problem is therefore in NP.

Some problems are not in NP. These are really theoretical problems.

The existence of a certificate implies most real life decision problems are in NP.

Reductions What does it mean for a problem to be "harder" than another problem?

Black Box: An algorithm which solves the problem. We don't know its run-time.

A problem A is easier than B $\iff$ a black-box for B could be called at most polynomial number of times to solve A. B itself may or may not take poly-time.

We say that A is polynomial-reducible to B, or that "A poly-time reduces to B",

$$A \leq_P B.$$

Proposition 18. *Polynomials are closed under multiplication and composition.*

Listing of NP Problems

"Easy" Problems (poly)	"Hard Problems"(NP Complete)
MST	Degree Bounded MST[2]
2-SAT	3-SAT
Eulerian Circuit	Hamiltonian Cycle
Min Cut	Max Cut

The vertical divider represents the poly-time boundary.

> **Theorem 19.** *If problem X in NP, then $X \leq_P SAT$. (Cook)*

i.e. SAT is the hardest problem in NP. The proof uses Turing Machines.

If SAT $\leq_P$ X (i.e. SAT can be poly-reduced to X) and X $\in$ NP, then X is NP Complete.

Hamiltonian Cycles, Max Cut are NP Complete.

Is SAT in NP? If so, then P=NP.

Lecture 8: Reductions

Transforming Optimization Problem to Decision Problem Suppose we want to maximize a function $f(x)$. We ask: is there an x such that $f(x) \geq t$?

Suppose we want to minimize a function $f(x)$. We ask: is there an x such that $f(x) \leq t$?

Once answered, we perform a binary search to find the optimal value. This takes logarithmic time with respect to the discretized unit of value.

Reductions They are transitive.

$$\text{If } A \leq_P B \text{ and } B \leq_P C, \text{ then } A \leq_P C.$$

This follows from polynomials form a closed set regardless of combinations.

NP-Complete For all $X \in NP$, $X \leq_P SAT$.

Recall, 2-SAT. e.g. $(X_1 \vee X_5) \wedge (X_3 \vee \neg X_4) \wedge \ldots$ with 2-literals per clause.

3-SAT has three literals per clause:

$$(X_1 \vee X_2 \wedge X_7) \wedge (\cdot\cdot\cdot)\ldots$$

NP-Hard Problems for which having a black box for them could allow us to solve SAT.

NP-Complete $X \in NP$ and X is NP-Hard.

Problems can be not in NP but still be NP-Hard.

We talk about NP-Hard for non-decision problems.

> If asked to show that a problem is NP-Hard, show that it can be used to solve SAT or any other NP-Complete problem

SAT has an arbitrary number of literals and clauses. We can transform any problem into 3-SAT such that the storage requirement is not exponentially large.

We now show that 3-SAT $\leq_P$ Max-Independent Set.

Theorem 20. *3-SAT $\leq_P$ Max-Independent Set.*

We are given a black-box for Max-Independent Set (i.e. we know how large an independent set the graph has).

Example input: $(X_1 \vee \neg X_2 \vee X_7) \wedge (\neg X_7 \vee X_2 \vee X_1)$.

Each variable is either `T` or `F`. Our black box either picks a node or it doesn't. How can we relate variables to nodes? Can I make it such that no node requires a variable to be both `T` and `F`?

For each clause, we make a triangle called a `Gadget`. We have k-clauses and k-gadgets.

Proposition 21. *Any independent set cannot have more than k clauses.*

(If more than k, we have a contradiction, since each one must come from a gadget, of which there are ...)

If Max Ind. Set returns us a node, we toggle it's value "on"/"off". To make sure that Max Ind. Set does not pick X_i and $\neg X_i$ to both be "on", we simply add an edge between X_i and $\neg X_i$ (for $i = 1, 2, \ldots, n$) such that they can never be returned by our independent set. (i.e. each variable may correspond to several nodes in our graph)

If we find an Ind. Set of size K (i.e. we can find one node from each gadget) that would satisfy a node from every clause ...

The structure of the graph forces nodes to picked from different gadgets.

Proposition 22. *A satisfying 3-SAT assignment implies there exists an independent set of size K.*

Proposition 23. *In the graph with negations connected, there exists an independent set of size k $\iff$ the formula is satisfiable.*

Proof. Proposition 22 and 21 yield the above claim. □

If we could use cliques instead of triangles, we could solve arbitrary K-SAT.

From Cook, we know that Ind Set $\leq_p$ SAT. Hence 3-SAT $\leq_p$ Ind. Set $\leq SAT$. Hence Ind. Set $=_p$ SAT. Hence independent set NP hard.

Vertex Cover "Where to put guards on nodes such that they can view all roads (edges)".

Given a graph $G(V, E)$, a vertex cover is a set $S \subseteq V$ such that all edges are incident to at least one node in S.

A trivial vertex cover is always possible using all nodes.

Minimum Vertex Cover What is the size of the smallest vertex cover? (rather, does there exist a smallest vertex cover of size k) This is NP-Hard.

Proposition 24. *Minimum Vertex Cover is NP Hard.*

Proposition 25. *Max Independent Set $\leq_p$ Minimum Vertex Cover.*

Theorem 26. *For any graph G, $S \subseteq V$ is an independent set $\iff$ $V - S$ is a vertex cover.*

Proof. $\longrightarrow$) If S is an independent set, no edges are in S. Hence any edges of G must be in $V - S$ or connected to at least one node in $V - S$.

$\longleftarrow$) Given we have a vertex cover, $V - S$. If there exists an edge strictly in S, then $V - S$ is not a vertex cover. Hence no edge contains two points in S. Hence S independent set. □

Assume that we have a black-box algorithm for Minimum Vertex Cover. Then the largest Independent Set is trivially the complement.

Hence Maximum Independent Set $\leq_p$ Minimum Vertex Cover.

Now, assume that we have a black box for Maximum Independent Set. From that, we take the complement to find the Minimum Vertex Cover.

Hence Max Independent Set $\geq_p$ Minimum Vertex Cover.

Whence, Max Independent Set $=_p$ Minimum Vertex Cover.

Further, Maximum Independent Set is NP-Complete because we related it to SAT. So we see that Minimum Vertex Cover is also NP Hard.

Lecture 9: Approximation Algorithms

Overview Now we move toward Approximation Algorithms. They relate to optimization problems which are NP-Hard. However, we can get a polynomial time algorithm to within ϵ of OPT.

- Turn optimization problems into decision problems to prove a problem is NP-Hard.
- Call optimum value for optimization problem "OPT" (OPT $\in \mathbb{R}$ since we're solving a max/min problem).
- The ratio between what approx. algorithm returns and OPT is called the "Approximation Ratio".

For Maximization problems, ratio < 1.

For Minimization problems, ratio > 1.

Randomized Max-Cut Approx for Max Cut, we can place each vertex in a cut by flipping a coin, whence $E[\text{Cut}] = m/2$; further we know that $OPT \leq m$. Then, we can say that $E[\text{cut/OPT}] \geq 1/2$. Hence we have a 1/2 approximation to an NP Hard problem.

Deterministic Max-Cut Approx Start with any cut. Check through all the nodes and ask, "does removing or adding this vertex to the cut increase cut size?" When this terminates, we realize a local maximum.

Algorithm 9: Max-Cut Approx (Deterministic)

```
Choose S ⊆ V arbitrarily (can start with ∅)
while there exists a vertex which can be swapped into (out of) our cut
    to achieve a higher-cut value do
    | Swap v to the "other side" of our cut S.
end
```

Proposition 27. *The algorithm terminates in polynomial time.*

Proof. Each "swap" increases the size of the cut by at least 1. The size of the cut is at most $|E|$. To check if there is a vertex which could be swapped to increase the Cut Size, we check $O(n)$ vertices hence each check terminates in finite time. Hence we have polynomial work being done inside a loop which has at most a polynomial number of iterations. □

Proposition 28. *Approximation ratio is* 1/2.

Proof. Note that OPT $\leq m$. Let C denote the output of our algorithm. Note that *at least* 1/2 the edges leaving *each* vertex must go across the cut (if not, we could move the vertex into (out of) the cut to get a larger cut size). i.e. for all nodes v, at least half the edges incident to v are cut. Hence,

$$C \geq \frac{1}{2} \cdot \sum_{i=1}^{n} \frac{\deg(v_i)}{2} = \frac{1}{4} \sum_{i=1}^{n} \deg(v_i) = \frac{2m}{4} = m/2.$$

Hence $\frac{C}{\text{OPT}} \geq \frac{m/2}{m} = \frac{1}{2}$. □

It's interesting to note that the best approximation algorithm for Max-Cut was $m/2$ for 45 years. In '95, Gomens and Williamson found a 0.878 approximation algorithm. We also know that if we can approximate Max-Cut better than 0.94, then $P = NP$. This is under the *Unique Games Conjecture*.

Vertex Cover Want to find the smallest subset of nodes such that all edges are covered.

Greedy Approach: take node with the highest degree. Remove this node and all incident edges. Approximation ratio is $O(\log n)$, which is un-boundedly bad since it grows with n.

Linear Program Approach: Let $X_i \in \{0,1\}_{i=1,2,\ldots,n}$ denote whether node i has been selected in the cover. The objective function is an affine function of indicators. We seek to solve

$$OPT = \min \sum_{i=1}^{n} X_i \text{ such that } \forall (i,j) \in E \qquad \underbrace{x_i \vee x_j}_{\text{i.e.} x_i + x_j \geq 1}$$

$$x_i \in \{0,1\} \qquad \text{for all } i = 1, 2, \ldots, n.$$

Realize that in this set-up, we want $x_i + x_j \geq 1$. Notice that the constraint $x_i \in \{0,1\}$ for all $i = 1, 2, \ldots, n$ is *not linear*. It's an NP hard problem.

But this is an exact problem. Let's relax our set-up.

We instead replace the constraint that $X_i \in \{0,1\}$ for $i = 1, 2, \ldots n$ with the relaxation that $x_i \in [0,1]$ for all $i = 1, 2, \ldots, n$.

Note that this solution, call it LP, is a relaxation of our original problem. Hence $LP \leq OPT$.

Algorithm 10: Minimum Vertex Cover

```
Solve LP specified above.
if x_i^* ≥ 1/2 then
    | x_i ← 1
end
else if x_i^* < 1/2 then
    | x_i ← 0.
end
```

Proposition 29. *After rounding, we still have a vertex cover.*

Proof. When we solve LP, we still respect that $x_i + x_j \geq 1$ for all $(i,j) \in E$. Hence at least one of x_i or x_j is at least 1/2. So when we do this rounding, we keep at least one node. □

We started with LP $\leq$ OPT. But with rounding, we've got a worse solution. How does it compare with OPT?

Proposition 30. *Our Approx Ratio is at most 2.*

Proof. We know that $\sum_{i=1}^{n} X_i^* = LP \leq OPT$. In the worst case, all x_i^*'s are 1/2. Hence all x_i's are 1.

Realize that $x_i \leq 2x_i^*$, since if $x_i^* \in [0, 0.5)$ then $x_i = 0 < x_i^*$. If $x_i^* \in [0.5, 1]$, then $x_i = 1 \leq 2x_i^*$. Hence

$$\text{Size of our Cover} = \sum_{i=1}^{n} X_i \leq 2 \sum_{i=1}^{n} X_i^* \leq 2\text{OPT}.$$

So, our approximation ratio is at most 2. □

Under Unique Games Conjecture, 2 is optimal. Without it, 1.36 optimal.

Bin Packing "You are only allowed to bring 2 suitcases. How much can we pack?"

Given n items, $a_1, a_2, \ldots, a_n \in (0,1]$. Find the minimum # of unit size bins to pack all a_i's.

This (as a decision problem) is NP hard. It's also very practical. E.g. we want to minimize the number of machines such that all jobs complete in under 1 minute.

Algorithm 11: Any Fit

```
for each a_i do
    if a_i fits in any bin then
        | Place it there.
    end
    else
        | open new bin.
        | place a_i in it.
    end
end
```

We claim this is a 2-approximation.

$$OPT \geq \sum_{i=1}^{n} a_i, \tag{3}$$

with equality if we could perfectly pack each unit bin to full capacity.

We have a Loop Invariant When we're done, there cannot be more than 1 bin which is 1/2 full. If this were true, we get a contradiction, since then our algorithm could have combined them into 1 bin.

"All bins are more than 1/2 full (except for at most 1 bin)". We prove this via induction. The base case is trivial.

Proof. If $a_i > 1/2$. It doesn't matter if we open a new bin, because if we do open a new bin, it will be more than 1/2 full.

If $a_i \leq 1/2$. There may or may not be an existing "shallow" bin (whose more than 1/2 free). If there is, we put a_i in it. Hence we end up with no shallow bins (loop invariant holds). If there isn't a shallow bin, we're allowed to open a new bin without violating the loop invariant. In all cases, the loop invariant holds. □

Let B denote the # of bins our Any-Fit algorithm returns.

Realize that the sum of the weights of the items must equal the sum of the loads of the bins (since we have to place each item somewhere). Hence, $\sum_{k=1}^{B} L_k = \sum_{i=1}^{n} \alpha_i \leq \text{OPT}$. By our above claim, B has at most 1 bin which is less than 1/2 full, hence

$$\frac{B-1}{2} < \frac{B-1}{2} + \underbrace{L_B}_{>0} \leq \sum_{k=1}^{B-1} \underbrace{L_k}_{\geq 1/2} + L_B \leq \text{OPT}$$

If we have B bins, $B-1$ are full. Hence $\sum_{i=1}^{n} a_i > \frac{(B-1)}{n}$. Using this fact and 3 we see that $B - 1 < 2\text{OPT}$.

Note that B and OPT are integers. Given strict inequality, adding 1 may make it an equality. Hence $B \leq 2\text{OPT}$.

Lecture: TSP, Dynamic Programming

Topics Covered TSP, Dynamic Programming (Fibonacci numbers, max window, exact exponential time TSP).

DP $\approx$ "Careful brute force"
DP $\approx$ guessing + recursion + memoization
DP $\approx$ shortest paths in some DAG

$$\text{time} = \#\text{subproblems} \times \underbrace{\text{time per subproblem}}_{\text{treating recursive calls as } \Theta(1)}.$$

Knapsack You're going camping. You can only afford to bring 1 giant back-pack. Suppose the size of the pack is the only thing that matters (can reformulate as weight). We can't bring everything (although that might be nice).

We have a list of items. Each with a *size* $s_i \in \mathbb{Z}$ and a *value* v_i. We have a knapsack total size $\mathcal{S}$. We want to choose a sub-set of the times (all if possible) whose total size is less than or equal to S such that we maximize sum of values.

(1) Sub-problems. Suffixes of items, $s[i:]$. So, starting with the ith item, we decide: do we include item i? We also track remaining capacity $X \leq \mathcal{S}$, for some integer X. Number of sub-problems is number of items times $\mathcal{S}$, i.e. $\Theta(n \cdot \mathcal{S})$.

(2) Guessing. We guess whether i is in our subset or not. There are only two choices.

(3) Recurse

$$\text{DP}(i, X) = \max\{\text{DP}(i+1, X), \text{DP}(i+1, X - s_i) + v_i\}.$$

(4) There is a topological order. We need constant time to evaluate each call to DP(,). Time is $\Theta(n \cdot \mathcal{S})$. This is *not* polynomial time in its current form. But $\mathcal{S}$ usually small. Hence this is *pseudo-polynomial*.

Symmetric Traveling Salesman Problem Given a weighted graph G with positive weights, find a Hamiltonian Cycle of minimum weight.

Even deciding *if* a Hamiltonian Cycle exists is NP Hard. Hence TSP (the full optimization problem) is also NP hard.

We cannot approximate this. So, suppose G complete with $\binom{n}{2}$ vertices. Hence any permutation of nodes is a Hamiltonian Cycle.

Even with a complete graph G, this is also NP hard. (For missing edges, we could just add them in with almost infinite weight, and translate any graph into a "complete" graph by which we run TSP)

Metric TSP We start with the assumption that all edge-weights satisfy $c_{ij} + c_{jk} \geq c_{ik}$, $\forall i, j, k$, where $c_{ij} = c_{ji}$ because this is *symmetric* traveling salesman problem. This prohibits adding infinite weight edges.

In order to satisfy this condition, all edges must be present, i.e. we have a complete graph. This is Metric TSP. We can approximate OPT to within 1.5x.

Naive Solution: Consider all permutations of nodes, each time realizing a Hamiltonian Cycle. Output the smallest weight cycle. Run time: $(n!) = \Theta(n^n)$.

Dynamic Programming: Exact solution in $O(n^2 2^n)$. This is possible for $n \approx 20$, e.g. when an Amazon robot is fetching items in the warehouse for your order. This is the problem we solve everyday when running errands.

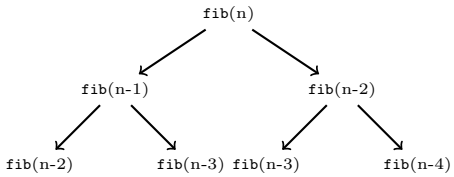
Dynamic Programming "While you write down a table, you figure out what you're going to write down next."

Fibonacci: $f_0 = f_1 = 1; f_n = f_{n-1} + f_{n-2}$.

Algorithm 12: Naive Fibonacci

```
1 if n == 0 or 1 then
2 |  return 1
3 end
4 else
5 |  return fib(n-1) + fib(n-2)
6 end
```

This has exponential run-time. Why?



We end up re-computing `fib(x)` over and over for each particular x. Hence we use memoization: we cache function values to avoid unnecessary recursion.

Algorithm 13: Fibonacci: Memoized

```
1 if n = 0 or 1 then
2 |  return 1
3 end
4 if computed(n) then
5 |  return cached(n)
6 end
7 newval ← fib(n-1) + fib(n-2)
8 cached(n) = newval
9 computed(n) = true
10 return newval
```

Now we simply wrap this function,

Algorithm 14: Fibonacci: memoized (wrapper)

```
1 Input: n, a non-negative integer.
2 cached(n) = array(n)
3 computed(n) = array(n) // all false
4 fib(n)
```

Here, we avoid recomputation. Specifically, we only call `fib(·)` at most n times. Hence the run-time is $O(n)$. The storage requirement is also $O(n)$.

This is the essence of dynamic programming: we take a problem, make it into a smaller sub-problem, and then remember (memoize) the values as we go.

Tangent: We can actually compute `fib(·)` even faster. Let,

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \qquad \text{then} \quad A^k \begin{bmatrix} f_{n-1} \\ f_{n-2} \end{bmatrix} = \begin{bmatrix} f_n \\ f_{n-1} \end{bmatrix}$$

We can diagonalize A and compute higher `fib(·)` in logarithmic time.

Max Sum Window Given an array, find the window of maximum sum. e.g.

$$[3 \quad -5 \quad 98 \quad -37 \quad 2 \quad 38 \quad 3 \quad 1 \quad 50]$$

Naively, we iterate through all $O(n^2)$ valid start and end indices and for each, compute sum in $O(n)$ time. Hence $O(n^3)$.

Let's reframe this via DP. Let `best(i)` = the size of the largest-sum window *ending* at i. That is,

$$\texttt{best}(i) = \max\{a_i, \texttt{best}(i-1) + a_i\}.$$

The original problem's solution is then $\max_i \texttt{best}(i)$. The total run-time is simply $2n$, since calculating `best(n)` requires n operations (and in the process, we memoize and compute `best(n-1)`, etc), and then we take the max over all $i = 1, 2, \ldots, n$.

Boilerplate

Analysis - How much possible work is done inside each recursive call? How many times do we have to call the function?

Backtracking - Our algorithm only returns the value, not where the index starts. To do this, we simply carry around another array and at each step, we track our maximizing (or minimizing) decision path.

TSP Given a graph with positive weights, we label the nodes arbitrarily $1, 2, \ldots, n$. Without loss of generality, start at node 1 (it's a cycle, so we can start anywhere). We have $n-1$ options of which node to visit next.

Algorithm 15: Dynamic Programming TSP

```
1 Input: S, a subset of vertices, and i, a node label.
  /* S ⊆ V, 1 ∉ S, i ∉ S                                  */
  /* Shortest path from 1 to i, using exactly each node in S.  */
2 TSP: min_i{DPTSP(V/{1, i}) + w(e_i1)}.
```

That is, we recurse through all neighbors i of node 1, each time picking the best way to form a (low-cost) cycle which starts at node i and uses all *yet to be used* nodes.

Hence,

$$\texttt{DPTSP}(S, j) = \min_{d_{ij}}\{\texttt{DPTSP}(S \backslash \{i\}, i) + d_{ij}\}.$$

Note that j is an argument to the function, hence constant over the minimization $\min_{d_{ij}}$. We make $O(n)$ recurrence calls since we minimize over all i. Each call is assumed to take constant time, since we memoize.

This takes exponential time. This solution doesn't require metric inequality to be satisfied; we don't even need symmetry.

What's the run-time?

$$\underbrace{2^{n-2}}_{\text{all possible } S} \cdot \underbrace{(n-1)}_{\substack{\text{all possible j} \\ \text{except 1}}} \cdot \underbrace{O(n)}_{\text{minimize over } i} = O(2^n n^2)$$

The space requirement is also $O(2^n n^2)$.

This is tolerable for about $n \approx 20$.

SAT How much time does naive SAT take? We are given n literals. The trivial solution is to toggle each literal, costing $O(2^n)$.

We've found $O(1.3^n)$ for SAT problem. This is exponentially faster than 2^n, and yet still exponentially slow.

Midterm Review

Basic Graph Theory

Trees and their characterizations. In particular, the proof that $m = n - 1$. That style of induction (via picking out the leaves).

Minimal Spanning Tree. Know Kruskal's algorithm and proof of correctness.

Matchings. No node incident on more than one edge. Perfect if every node matched exactly 1x.

Independent Set/Clique ↔ Vertex Cover. The problems go hand in hand, since every time you yield a cover, we get a bound on the size of the matching. Recall the proof that minimum vertex cover is NP Complete.

Hamiltonian Cycles, Eularian Tours.

Diameter, radius, shortest paths.

Cuts Global Min-Cut: 1 value, $\binom{n}{2}$ cuts which realize this value. s-t Min Cuts: for fixed s and t, there is one minimum cut-value. For all s and t, there are only $n-1$ distinct values. For fixed s and t, there may be exponentially many cuts which realize the minimum cut value.

The Max Cut problem is NP Hard. Randomized algorithm and Deterministic algorithm: 1/2.

Min-Cut/Max-Flow Applications Integer capacities imply integer max flows. Hence if the capacity is 1, the flow is either "on" or "off", hence we can do things like edge disjoint paths.

Recipe.
(1) Is it a maximization or minimization problem? i.e. a flow or a cut.
(2) What graph to use? i.e. how to code up the constraints. Generally, there is 1 node per inequality. We use an upper bound on out-capacity.
(3) How to recover the solution? Usually the max-flow or min-cut value. Sometimes, the paths in the residual graph i.e. which edges are saturated.

Cover Times

$$c_{ij} = 2mR_{ij}.$$

From that, using a spanning tree and effective resistances between a single edge being at most 1 ohm, we see that

$$C(G) \leq 2m(n-1).$$

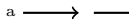
This is useful when the graph is sparse. If the true cover time is less than $O(n^2)$, this bound isn't useful, e.g. for a connected graph.

$$(C) \leq 2me^3 \ln nR(G) + n = O(m \ln nR(G)).$$

Cover times trivially give a bound on commute and hitting time, since

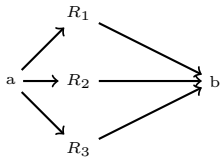
$$mR(G) \leq C(G).$$

We have series circuits, whose effective resistances are given by the a-b path length,



and parallel circuits, whose effective resistance is given by

$$R_{ab} = \frac{1}{\sum_{a\text{-}bpaths} \frac{1}{\text{path length}}},$$



When asked to calculate $C(G)$, i.e. $\Theta(C(G))$, need to show that $C(G) \leq \cdot$ and $\cdot \leq C(G)$.

To figure out how many paths there are between a and b and if they are saturated, hence we use Max Flow and Min Cut.

Adding an Edge cannot increase $R(G)$ To see this, . . .

Adding an edge doesn't tell us anything about Cover Time. Consider . . .

Probabalistic Method Union Bound, Ramsey Theorem, Markov Inequality, Basic Counting, Expectations of Indicator Variables.

Showing $\Pr(x) > 0 \implies x$ exists.

NP-Hard Problems If a problem is in NP and it is NP-complete, it is NP-Hard. If we are given a new problem, and it can be used to solve an NP-Hard problem, then it's NP-Hard.

Integer Programs. LP with 0-1 constraints. Integer programs are NP-Complete.

Minimum Vertex Cover (2-approximation via LP)

Bin Packing (2-approx)

Maximum Independent Set

Maximum Cut (1/2 approximation, deterministic and randomized)

Traveling Salesman (often assume Metric Inequality)

SAT (and 3-SAT)

Know how to form optimization problem into a decision problem, and then talk about whether it's NP-Hard. I.e. to prove that optimization problem NP-Hard, we first turn the problem into a decision problem (often by adding a threshold to the problem definition); use that as a black-box algorithm to solve our NP-Complete problem.

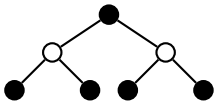
Dynamic Programming Exponential time solution to TSP. Fibonacci Numbers, Maximum Window.

Last Years Midterm

#1) Every tree on n nodes has a vertex cover of size at most

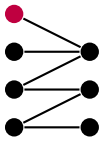
$$\left\lceil \frac{n-1}{2} \right\rceil.$$

Failed attempt: there are n-1 edges, pick one side of every edge to be in the vertex cover. This doesn't work. To see this, consider the following graph:



Define the layers in this graph ℓ_i, for $i = 1, 2, 3$. Start at the root node. Our algorithm traverses through a list of all edges. It has two options: for each edge, use the parent node for the vertex cover. Or, for each edge, use the child node for the vertex cover. The former yields vertex cover of size 3 (we choose nodes at ℓ_1, ℓ_2), Whereas yields cover of size 6 (we choose nodes at ℓ_2, ℓ_3). OPT here is to use only white nodes, for cover size 2. Hence even in this toy algorithm, we can end up using $n-1$ nodes, which is larger than the required $\lceil (n-1)/2 \rceil$.

A better solution: Notice that every tree is a bi-partite graph. We just start from the root, use BFS or DFS to explore nodes, and color vertices red or blue according to the parity of the "layer" in the tree (i.e., path length from node to the root of the tree). By definition, there are 2-sides in a bi-partite graph. There are two cases to consider, depending on n even or odd. In either case, min-vertex cover given by $\min\{|L|, |R|\} = \left\lfloor \frac{n}{2} \right\rfloor = \left\lceil \frac{n-1}{2} \right\rceil$.



If n even, $n/2$ a number hence floor/ceiling operator nil-potent. If n odd (including colored node) $n/2$ fractional, hence taking floor operator we round down. This is identical to taking the ceiling of $(n-1)/2 = n/2 - 1/2$.

Finally, note that in any bi-partite graph, each side is a vertex cover.

The max-cut in a bi-partite graph is m since we can cut all edges.

#2) Prove that every tree has at most 1 perfect matching.

Proof. Induct on the number of nodes on the tree. For $n = 1$, no matching exists. For $n = 2$, exactly one perfect matching exists. Assume that for all trees with $\leq k$ nodes, at most one perfect matching exist. Consider a tree on $k+1$ nodes. There exists some leaf node l, which in any perfect matching must be attached with its parent node (because that is the only edge incident to l). Delete l, its parent, and all incident edges to those nodes from the tree. We are left with a forest, in which every tree has $\leq k-1$ nodes. We know by the inductive hypothesis that each of these trees has at most one perfec matching, so the original tree has at most one perfect matching. This follows from the unique perfect matching of each tree in the forest and the edge connecting l to its parent. □

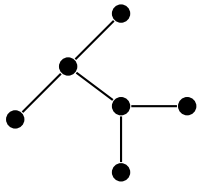
We note that any tree with an odd # of nodes can't have a perfect matching. Even if there is an even number of edges, there is not always going to be a perfect matching, e.g.

Approximation Algorithms

Overview The class now becomes focused on approximation algorithms which leads us to cutting-edge research. We focus on the Metric TSP, finding a 2-approximation using MST's and a 1.5 approximation which is state of the art. We also get a $\log n$ approximation to ATSP.

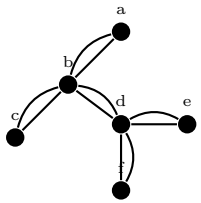
Metric TSP - 2 Approximation
We first need to get a handle on OPT. To do this, we use a MST. Examine a tour of the graph (a cycle). If we delete an edge, we get a spanning tree. Notice that the MST is the lowest weight tree. Hence MST $\leq$ OPT, since OPT is a spanning tree plus a non-negative weight edge to complete the tour.

Suppose we have an MST which looks as follows.



Notice that some vertices have odd degree, hence there does not exist an Eularian circuit.

Step #1 Turn the graph into a multi-graph. That is, we double each edge of the MST, to yield a graph



Since we doubled each edge, we double the weight of the MST.

Step #2 Find an Eularian Tour greedily. We do this by starting at an arbitrary node and greedily choosing unexplored edges to traverse next. We may realize a cycle, but then we simply backtrack to the first node with a not-yet-traversed outgoing edge. Such an edge must exist since the graph connected (it is a spanning tree, and then we added edges). Notice that in this Eularian Tour, some nodes are repeated. This prevents us from returning this as our output of a Hamiltonian Cycle.

How can we fix this issue of re-visiting nodes? Suppose our Eularian Tour is of the form

$$a \to b \to c \to b \to d \to f \to d \to e \to d \to b \to a.$$

Step #3. Notice that our original graph is complete. Hence we can simply apply a "shortcutting" method, where we take the above Eularian Tour, follow it in order until we are about to revisit a node for the first time. Instead of visiting the node for a second time, we simply take the direct edge to the following node which has yet to be visited. Such an edge exists since the graph is complete, and we incur no additional cost by the metric inequality. Our Hamiltonian Cycle then becomes

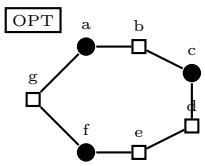
$$a \to b \to c \to \cancel{b} \to d \to f \to \cancel{d} \to e \to \cancel{d} \to \cancel{b} \to a.$$

Hence ALG $\leq 2 \cdot$ MST $\leq 2 \cdot$ OPT. Hence we have a 2-approximation.

Metric TSP - 1.5 Approximation Now notice that in our previous algorithm, we doubled each edge in our MST regardless of its degree. What if we only add edges for odd-degree nodes as needed? This could be more efficient.

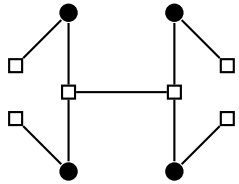
Notice that by the Handshake Lemma, $\sum_{i=1}^{n} \deg(v_i) = 2m$. Hence the number of odd-degree nodes must be even (for any graph). To see this, realize that the number of edges in the graph is given by $m = \frac{1}{2}\sum_{i=1}^{n} \deg(v_i)$, hence since m a non-negative integer, $\sum_v d(v)$ is an even number. From this sum, discard all vertices with even degree. If there are no vertices left, we're done. Otherwise, we have odd-degree vertices left. But since we started with an even number, and subtracted all vertices with even degree (i.e. subtracted another even number) we are left with an even number. Hence if we try to assert that there is only an odd-number of odd-degree vertices, we get a contradiction, since $\texttt{odd} \cdot \texttt{odd} \neq \texttt{even}$.

Suppose we are given an instance of OPT, where the hollow nodes denote odd-degree vertices.



where the previous graph has 2-perfect matchings among odd-degree nodes (one in dashed edges, one with solid bent edges).

If we wish to improve on our 2-approximation, we can't just start adding edges to odd-degree nodes since this just propogates the problem. For example, consider the spanning tree.



where again the vertices with odd-degree are in white-squares, and vertices of even-degree are in black-circles.

We seek to obtain a perfect-matching among odd-degree nodes. Intuitively, we want to add in edges cheaply from our original (complete) graph.

Blossom Algorithm: A generalization of the Residual Graph. It's a poly-time algorithm to compute min-weight perfect matching on any graph.

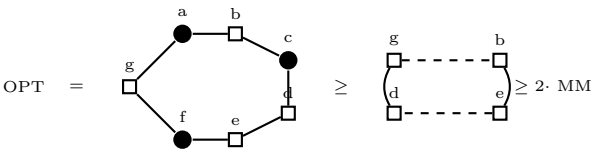
Algorithm 16: Christofedes 1.5x (Metric) TSP

1. Compute MST
2. Compute min-weight matching between odd-degree nodes (Blossom)
3. Compute Eularian Tour
4. Shortcut nodes to achieve Hamiltonian Path.

When we compute the min-weight matching between odd-degree nodes, we add exactly 1 degree to each odd-degree vertex. After which, all nodes have even degree, hence an Eularian tour exists.

Consider again our instance of OPT shown below. We now move directly between odd-degree nodes. Between each, we "skip" even-degree nodes (i.e. we shortcut). Notice that the cost of the resulting cycle we get is bounded above by OPT since we applied shortcutting. Further, the resulting cycle is even length since there are an even # of odd-degree nodes.

Now, we take every other edges in this resulting cycle, which yields a matching. We take a look at the other (unused) edges. This results in another perfect matching. Hence we have an ordering by weight:



where MM denotes the Maximum Matching.

If M a min-cost perfect matching on $V' \subseteq V$ such that $|V'|$ is even, then $\text{cost}(M) \leq \text{OPT}/2$. Consider an optimal TSP tour of G, call it τ. Let τ' be the tour obtained by short-cutting τ. By metric inequality, $\text{cost}(\tau') \leq \text{cost}(\tau)$. Realize that τ' is the union of two perfect matchings on V', each consisting of alternating edges of τ. The cheaper of these matchings has cost $\leq \text{cost}(\tau')/2 \leq OPT/2$. Hence optimal matching also has cost at most $OPT/2$. Hence MM $\leq \frac{1}{2}$ OPT.

By our first argument in the 2-approximation, we also have that MST $\leq$ OPT.

Hence Christofedes Algorithm $\leq \frac{3}{2}$ OPT.

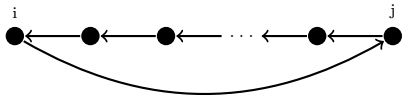
Metric Asymmetric TSP Notice that with for our Symmetric TSP problem, we assumed we could duplicate edges in an MST in effort to find an Eularian Tour (which we then shortcut). Now we're in a *directed* graph. We no longer have spanning trees (these aren't defined in a di-graph).

We still have a complete graph, and we still have the metric inequality, i.e. $\forall i, j, k\ c_{ij} + c_{jk} \geq c_{ik}$. However, we are in a directed graph. Taversing one way is much more expensive.

Example: Construct a graph G with the following distances. Let δ_{ij} denote the shortest-path between i and j in some auxiliary graph H, which is unweighted. By definition shortest-paths, the δ_{ij}'s must

satisfy the triangle-inequality (i.e. they are metric). H has the same nodes as G. We want to construct δ_{ij} very different from δ_{ji}.

Take an undirected path from i to j, and add a directed edge from j to i.



Hence $\delta_{ij} = 1$ and $\delta_{ji} = n - 1$.

More generally, we consider a directed cycle containing all nodes.

Cycle Covers: Given any (un)directed graph, a cycle cover is a collection of cycles which cover all nodes.

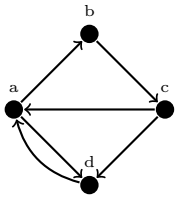
This may not exist, e.g. a tree, or a lollipop.

Is there a cycle cover in a graph?

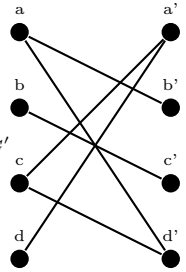
Algorithm 17: Detect Cycle Cover

1 Turn directed graph into bi-partite graph with $2\times$ # nodes.
2 There is a cycle cover $\iff$ a perfect matching exists in bi-partite graph.

For example, given a graph G,



we construct a graph G'



So, our cycle cover exists in $G \iff$ a perfect matching exists in G'. We will never re-use nodes since a perfect matching ensures each node is incident to exactly one edge in the matching.

We take the following theorem for granted.

Theorem 31. *There exists a perfect matching in $G' \iff$ there exists a cycle cover in G.*

Now, we apply the Blossom Algorithm to get the minimum weight perfect matching. Hence,

In directed graphs, we can compute minimum cycle covers.

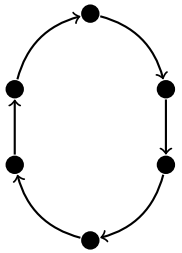
That is, relaxing the problem of finding a Hamiltonian Cycle (i.e. a cycle cover consisting of only 1 cycle) to finding a cycle cover (more than one cycle) makes the problem tractable.

Let the ATSP optimum be denoted as OPT. The weight of the Min-Cycle Cover (MCC) is at least as small as Hamiltonian Cycle since Hamiltonian Cycle is a special case of Cycle Cover. Hence MCC $\leq$ OPT.

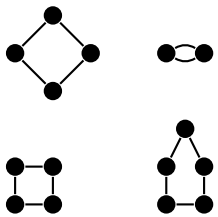
Intuitively, we seek to massage the MCC into a Hamiltonian Cycle without increasing the weight too much.

Suppose OPT and MCC look like

OPT



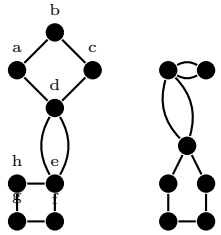
MCC



How do we connect the cycles without adding too much weight? We use another cycle cover! In an MCC, the # connected components is $\leq n/2$. To see this, notice that each node connected to at least one other node.

Algorithm 18: ATSP $\log n$ Approximation

1 Compute Cycle Cover.
2 Pick one node (representative node) from each cycle.
`// This reduces the # CC's by at least 1/2. We may have not-simple cycles in the result.`
3 Compute MCC among representative nodes.
`// Resulting MCC is on a subset of nodes of` G`, hence we can shortcut it a cycle cover, and recurse.`
4 Resulting graph has all even-degree. Construct an Eulerian Circuit, and shortcut it.
5 Recurse.



So, we have an Eulerian Tour

$$a \to b \to c \to d \to e \to f \to g \to h \to e \to d \to a$$

which we shortcut to yield a simple cycle (this is where triangle inequality important),

$$a \to b \to c \to d \to e \to f \to g \to h \to \not{e} \to \not{d} \to a$$

We repeat our algorithm for $O(\log n)$ iterations, since each time we *half* the number of connected components.

The output of the algorithm is a simple cycle. The total cost is

$$\mathrm{MCC}_1 + \mathrm{MCC}_2 + \mathrm{MCC}_3 \ldots + \mathrm{MCC}_{\log n} \leq \log n \mathrm{MCC} \leq \log n \mathrm{OPT}.$$

Hence our approximation ratio is $\lg n$.

Arboresence: Starting from 1 node, can you get to any other node?

Max Cut We note that Linear Programs are convex. Semi-definite programs are also convex. The general idea is that we start with our problem, then relax it until we can solve a convex program. From there, we then massage the output to be feasible.

Semi-definite Program Here we write Max-Cut as a semi-definite program.

$$\begin{aligned} \max_{x_{ij}} \quad & \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij} x_{ij} \\ \text{subject to} \quad & [x_{ij}] \succeq 0. \end{aligned}$$

where $\succeq$ denotes that our matrix of x_{ij} entries is positive semi-definite, i.e. the eigenvalues are non-negative. Here X must be symmetric positive semi-definite.

Max Cut We start with a non-solvable program. Given $G(V, E)$ unweighted, we create one variable per node indicating whether it is "in the cut". That is, let

$$y_i \in \{-1, 1\}, \qquad i = 1, 2, \ldots, n.$$

To see if two variables are on different sides of the cut, we simply multiply the two corresponding variables. If nodes i and j are both in the cut, or both not in the cut, then $y_i \cdot y_j = (\pm 1)^2 = 1$. If i and j are on different sides of the cut, then $y_i \cdot y_j = -1$. With this in mind, we express our program as follows,

$$\begin{aligned} \max_{y_i} \quad & \sum_{(i,j)\in E} \frac{1 - y_i y_j}{2} \\ \text{subject to} \quad & y_i^2 = 1, \quad y_i \in \mathbb{R} \end{aligned}$$

Our summation now counts counts, since $y_i \in \{-1, 1\}$ and the expression

$$\frac{1 - y_i y_j}{2} = \begin{cases} 1 & \text{if } (i,j) \text{ is in the cut} \\ 0 & \text{if nodes are on same side of cut.} \end{cases}$$

Hence our above program is a legitimate Max Cut using n variables. We now must apply a relaxation to turn our problem into a semi-definite program. Following Goemans and Williamson.

Relaxation: Allow each y_i to be in $\mathbb{R}^n$, call the resulting vector v_i. Now, the number of variables is n^2. Now, our program is

$$\begin{aligned} \max_{y_i} \quad & \sum_{(i,j)\in E} 1 - v_i^T v_j \\ \text{subject to} \quad & \|v_i\|_2^2 = v_i^T v_i = 1, \quad v_i \in \mathbb{R}^n. \end{aligned}$$

Let us rename $v_i^T v_j$ to x_{ij}, i.e. we store all inner products v_i, v_j into a symmetric matrix X. Now, we have SDOPT.

$$\begin{aligned} \max_{(i,j)\in E} \quad & \sum_{(i,j)\in E} \frac{1 - x_{ij}}{2} \\ \text{subject to} \quad & x_{ii} = 1, \qquad \forall i = 1, 2, \ldots, n, \\ & [x]_{ij} \succeq 0. \end{aligned} \tag{4}$$

From Linear Algebra, we know that

Theorem 32. $[x]_{ij} \succeq 0 \iff x_{ij} = v_i^T v_j$.

Clearly SDOPT $\geq$ OPT. This follows directly from the fact that our relaxation increases the feasible region (i.e. the possible set of inputs). At worst, we could achieve OPT.

Theorem 33. *A matrix A is symmetric positive definite $\iff$ there exists B such that $A = B^T B$, i.e. a* Cholesky Decomposition *exists. Further, for a symmetric $n \times n$ matrix, the following are equivalent,*

1. *There exists an $m \times n$ matrix V such that $V^T V = A$,*
2. *For all $x \in \mathbb{R}^n, x^T A x \geq 0$, and*
3. *All eigenvalues of A are non-negative.*

We now present the Fully Polynomial Time Approximation Solution (FPTAS).

Algorithm 19: Goemans-Williamson FPTAS (95)

1 Solve SDP.
2 Take output X, perform Cholesky to yield XV^TV.
`// Now, each node has associated with it a unit vector in` $\mathbb{R}^n$.
3 Take a random hyper-plane through the origin.

When we take a random hyper-plan through the origin, notice that

$$\Pr(\text{hyper-plane separates two vectors}) = 2 \cdot \frac{\text{angle between vectors}}{360}.$$

Recall that

$$v_1^T v_2 = \|v_1\|_2 \|v_2\|_2 \cos\theta.$$

Hence

$$\theta = \cos^{-1}(v_1^T v_2) = \cos^{-1}(x_{12}).$$

That is, $\Pr(\text{edge}(1,2)\text{cut}) = \frac{\cos^{-1}(x_{12})}{\pi}$.

Now,

$$E[\text{cut}] = \sum_{(i,j)\in E} \Pr[(i,j) \text{ is in the cut}] = \sum_{(i,j)\in E} \cos^{-1}(x_{ij})/\pi.$$

Recall that since SDOPT $\geq$ OPT (since SDOPT a relaxation) our approximation ratio is given by,

$$\text{Approx. Ratio} = \frac{E[\text{cut}]}{\texttt{OPT}} \geq \frac{E[\text{cut}]}{\texttt{SDOPT}} = \frac{\sum_{(i,j)\in E} \cos^{-1}(x_{ij})/\pi}{\sum_{(i,j)\in E} \frac{(1-x_{ij})}{2}}$$

$$\geq \min_{-1\leq x\leq 1} \frac{\frac{\cos^{-1}(x)}{\pi}}{\frac{1-x}{2}} = 0.878.$$

To get to the last step, we minimize via calculus to get a lower bound.

Concentration Inequalities

Overview Today, we review Held-Karp LP for TSp. We discuss Concentration Inequalities. We also start Sparsification and matrix-graph connections.

TSP LP We wish to turn the TSP into one of selecting an ordering of edge weights.

Objective: minimize the tour's length.

We create a 0-1 variable for each edge, which allows us to write

$$\begin{aligned} \min \quad & \sum_{(i,j)\in E} x_{ij}\cdot c_{ij} \\ \text{subject to} \quad & x_{ij}\in\{0,1\}, \\ & \sum_{j:(i,j)\in E} x_{ij} = 1 \iff \begin{cases}\text{out-degree}(i) & =1\\ \text{in-degree}(i) & =1,\end{cases} \\ & \forall S\subseteq V, \text{out-degree}(S)\geq 1. \end{aligned} \tag{5}$$

We require each node to be visited, i.e. the out-degree and the in-degree of each node must be 1. Hence we impose our second constraint. At this point, we get a Min-Cycle Cover. To ensure connectivity, we require that for each cut $S\subseteq V$, there is at least an outgoing edge.

Now we have an LP for an NP-Hard problem.

Relaxation:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in E} x_{ij}\cdot c_{ij} \\ \text{subject to} \quad & \underbrace{x_{ij}\in[0,1]}_{\text{relaxation}}. \end{aligned}$$

The problem with this is that we have *exponentially many* constraints.

Karp-Dimitron: Linear Programs with exponentially many constraints can be solved in Poly-time as long as there exists an Oracle for finding violated constrains in a poly-type.

To do this, we find the Global Min Cut.

If the value of the Cut < 1, the graph is disconnected. Hence we have violated a constraint.

If the value of the cut ≥ 1, the constraints are satisfied.

1. Solve HK-LP.
2. Saberi et al, approximation factor is $\log n/\log\log n$.

Sparsification

Graph/Matrix Connections We define an *adjacency matrix*

$$A = \left[a_{ij}\right], \qquad a_{ij} = \mathbf{1}\{(i,j)\in E\}.$$

Proposition 34. *A has the property that we can use it to calculate the # of paths of a certain length. That is,*

$$\left[A^k\right]_{ij} = \#\text{ of paths of length exactly k between i and j},$$

where the paths are not simple (i.e. they are allowed to revisit nodes).

Proof.

$$\left[A^2\right]_{ij} = \sum_{k=1}^{n} A_{ik}A_{kj} = 1 \text{ for each path length of 2,}$$

i.e. we are counting paths. That's really the inductive step. The base case is A^2. □

Random Walk Matrix Suppose W is a transition matrix of probabilities, i.e. a *Stochastic Matrix*.

$$W = D^{-1}A,$$

$$D = \text{matrix of degrees} = \begin{bmatrix}\deg V_1 & & & \\ & \deg V_2 & & \\ & & \ddots & \\ & & & \deg V_n\end{bmatrix}$$

This assumes we have a connected graph, i.e. $\delta(G)\geq 1$. To get around this, we may add in self-loops (recall from Stats 217, making a Markov Chain lazy).

$$\left[W^k\right]_{ij} = \text{probability of landing at } j \text{ starting from } i \text{ in exactly k steps}$$

Recall that multiplying on the left by a diagonal matrix scales the rows of A according to the entries of D. Hence each row of W sums to 1, i.e. for each node it's a probability distribution among outgoing edges.

Laplacian Matrix We deal with simple, undirected, unweighted, connected graphs. For a symmetrix matrix, i.e. an undirected graph

Definition 1:

$$L = D - A \qquad n\times n \text{ matrix.}$$

We can now count cuts with L.

Definition 2: Take an undirected graph G and arbitrarily orient the edges (i.e. we add exactly one arrow to each edge). We construct B, an $m\times n$ matrix where

$$B_{ev} = \begin{cases}1 & \text{if vertex v is head of e}\\ -1 & \text{if vertex v is tail of e}\\ 0 & \text{otherwise.}\end{cases}$$

Then, $L = B^TB$ is an $n\times n$ matrix.

> The spectrum of the Laplacian lets us understand how cuts are structured in our graph.

Let x_S be an indicator vector for the cut $S\subseteq V$, i.e. $x_S\in\{0,1\}^n$ where entry i takes on value 1 if node $i\in S$ and zero otherwise.

Theorem 35. $x_S^TLx_S = cut(S)$.

Proof.

$$x^TLx = x^TB^TBx = \|Bx\|_2^2$$

Each entry i above is calculated by B_i^Tx, where B_i^T has a single 1 where the head of the edge is and a single 1 where the tail of the edge is. Hence, for each entry $Bx\in\mathbb{R}^m$,

$$\|Bx\|_2^2 = \sum_{(i,j)\in E} \underbrace{(x_i - x_j)^2}$$

where

$$(x_i-x_j)^2 = \begin{cases}1 \text{ if } x_i - x_j\in\{-1,1\}\\ 0 \text{ if } x_i - x_j = 0.\end{cases}$$

Hence for each edge, we count: if x_ix_j spans the cut, we get a -1 or a 1, squared $\to$ 1, hence we count cuts. □

Theorem 36.

$$L\mathbf{1} = 0.$$

Proof.

$$L\mathbf{1} = (D-A)\mathbf{1} = \underbrace{D\mathbf{1}}_{\text{degree of each node}} - \underbrace{A\mathbf{1}}_{\text{degree of each node}} = 0.$$

□

It immediately follows that $\mathbf{1}$ is an eigen-vector with corresponding eigenvalue 0. Since for any matrix C, $\det(C) = \prod_i \lambda_C(i)$, we see that $\det(L) = 0$ and hence L singular.

Cuts and Spectral Sparsification Problem: Given an undirected, unweighted, simple, connected graph G with m edges, find weighted (undirected, simple, connected) H with few edges such that $\forall S\subseteq V$ (all cuts),

$$\text{Relative Error} = \frac{|f_G(S) - f_H(S)|}{f_G(S)} < \epsilon.$$

Here $f_G(\cdot)$ is an unweighted cut function (i.e. it simply counts # edges in the cut) and $f_H(S)$ is a weighted cut function (i.e. it sums the edge weights in the cut) with H having

$$O\left(\frac{n\log n}{\epsilon^2}\right) \text{ edges.}$$

We do this via effective resistances. This is a 2009 result. Batson, Spielman, and Srivistava further brought this down to $O\left(\frac{n}{\epsilon^2}\right)$ in 2011.

We first try something simple.

Algorithm 20: Naive Sparsification

```
for each edge e ∈ G do
    Flip a biased coin c. /* With probability p, coin lands heads. */
    if Heads then
        Include e in H with weight 1/p.
    end
end
```

Proposition 37. $f_H(S)$ *unbiased.*

Proof. For any $S\subseteq V$, consider

$$\begin{aligned} E\left[f_H(S)\right] &= \sum_{\substack{e\in E:\\ e \text{ leaving } S \text{ in } G}} [\Pr(\text{edge e selected})\cdot w(e)] \\ &= \sum_{\substack{e\in E:\\ e \text{ leaving } S \text{ in } G}} p\cdot\frac{1}{p} = \sum_{\substack{e\in E:\\ e \text{ leaving } S \text{ in } G}} 1 = f_G(S). \end{aligned}$$

□

Hence, in expectation our graph H is unbiased for cut-sizes.

Notice that as $p\to 1$, we are more likely to have $f_G(S) = f_H(S)$. But of course as $p\to 1$ we get more edges. We want to set p small, but if we do the results become very volatile.

Recall the Markov Inequality. If X is a non-negative Random Variable, then

$$a\cdot\mathbf{1}\{x\geq a\}\leq x \implies a\cdot E[\mathbf{1}\{x\geq a\}]\leq E[x] \implies \Pr(X\geq\alpha)\leq\frac{E[X]}{\alpha}.$$

Hence for $x\geq 0$,

$$\Pr(x\geq\alpha E[x])\leq\frac{E[x]}{\alpha E[x]} = \frac{1}{\alpha}.$$

We want something kind of like this. We are interested specifically in *Concentration Inequality*, hence we turn to the *Chernoff Bound*.

Chernoff Bound - Sum of indicator random variables which may or may not occur with the same probability. Let $X = \sum_{i=1}^n x_i$, where $x_i\in\{0,1\}$, $x_i\text{Bern}(p_i)$, and $\mu = E[X] = \sum_{i=1}^n E[x_i]$. Then,

$$\Pr(X\geq(1+\delta)\mu)\leq\exp\left[-\delta^2\mu/2\right]$$

$$\Pr(X\leq(1-\delta)\mu)\leq\exp\left[-\delta^2\mu/3\right].$$

Using a Union Bound,

$$\Pr(|X-\mu|\geq\delta\mu)\leq\exp\left[-\delta^2\mu/2\right]+\exp\left[-\delta^2\mu/3\right]\leq 2\exp\left[-\delta^2\mu/3\right].$$

Notice that for μ larger, we get more concentration around the mean.

E.g. Given n fair coins, toss them. We expect $n/2$ heads. As n grows, the probability of being δ-close to μ grows exponentially fast toward 1.

For cuts with more edges, the our naive procedure is not too volatile. But for cuts with very small number of edges, our approach is very volatile, since it depends on the outcome of a small number of coin-flips.

For fixed $S \subseteq V$, suppose $f_G(S) = c$. We have that (two steps needed?)

$$\Pr\left(\left|f_H(S) - c\right| \geq \underbrace{\epsilon c}_{=\delta\mu}\right) \leq 2\exp\left[\frac{-\epsilon^2 pc}{3}\right],$$

where $f_H(S)$ is the random variable. (how did we insert p above?)

This is the probability that we achieve our guarantee for sparsification.

The probability is small when ϵ^2 dominates.

If p large, the probability of deviation small. How large do we need to set p?

If $f_G(S)$ large, probability of deviation also small. This is not something we get to choose, however. It's also possible that $p \leq 1$ and $f_G(S) = 1$.

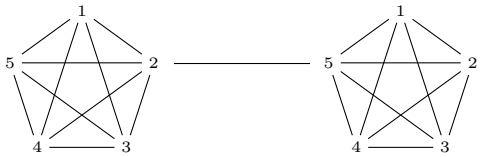
Suppose

$$p = \frac{3\log n}{\epsilon^2 \cdot c},$$

where c denotes the global minimum cut. Then,

$$\Pr\left(\left|f_H(S) - f_G(S)\right| \geq \epsilon f_G(S)\right) \leq 2\exp\left[-\log n\right] \leq \frac{2}{n}.$$

If $c \geq \log n$, we get nice results. But if $c \ll \log n$, this scheme fails. Consider a bar-bell graph, for example.



In this example, the edge connecting the two cliques is a 1 ohm resistor; it must be left in the graph in order for us to measure any cut-size correctly. So $c = 1$ in this example; Hence for the bar-bell graph $p = \log n/\epsilon^2 > 1$, which is not even a probability.

We abort our old attempt in favor of effective resistances, which get us the notion of how important an edge is for a cut.

Algorithm 21: Sparsification, Spielman-Srivastava (09)

```
Let q = O((n log n)/ε²).
for i = 1, 2, ..., q do
    Sample edge e with probability R_e/(n−1).
    H += e with weight 1/R_e.
end
```

where the normalizing constant for R_e is the sum of the effective resistances (which we show is equal to $n-1$, for any graph).

This defines a distribution over edges. For each of the q rounds, we choose an edge from a distribution based on the edge's effective resistance in a graph. If H doesn't yet contain edge e, we add it with weight $1/R_e$. If H does already contain e, we increase the weight of e in H by $1/R_e$, i.e. we "fatten" the edge.

We now compute Effective Resistances via Linear Algebra and the Laplacian. Recall B is an incidence matrix $\in \mathbb{R}^{m\times n}$, where

$$B_{e,v} := \begin{cases} 1 & \text{if } v \text{ is head of } e \\ -1 & \text{if } v \text{ is tail of } e \\ 0 & \text{otherwise.} \end{cases}$$

We define $L = B^T B$, where $L^\dagger$. We further define

$$\Pi = BL^\dagger B^T, \qquad \text{where } \Pi_{ee} = R_e.$$

Proposition 38. *We claim that span$\{\mathbf{1}\}$ = kernel(L) = kernel(B).*

Proof. Recall kernel(L) is the set of vectors which gets killed by L. It's clear that if $x \in \mathbb{R}^n$ such that $Bx = 0$, then $Lx = B^T Bx = B^T(0) = 0$, hence kernel$(B) \subseteq$ kernel(L).

What about the other way? Notice that

$$\begin{aligned} Lx = 0 \implies x^T Lx = 0 &\iff x^T B^T Bx = 0 \\ &\iff \|Bx\|_2^2 = 0 \\ &\iff Bx = 0 \\ &\iff \sum_{(i,j)\in E} (x_i - x_j)^2 = 0 \\ &\iff x_i = x_j \forall (i,j) \in E. \end{aligned}$$

Lastly, note that since G connected,

$$\forall (i,j) \in E \iff X \in \text{span}\{\mathbf{1}\}.$$

□

Also, recall that $L = (D - A)\mathbf{1}$. Hence via an eigen-value decomposition

$$L = \sum_{i=1} \lambda_i v_i v_i^T$$
$$L^\dagger = \sum_{\lambda_i \neq 0} \frac{1}{\lambda_i} v_i v_i^T,$$

where we note that exactly only 1 of the λ_i are zero since the span is $n-1$.

$LL^\dagger$ is a projection onto the space spanned by the v_i's.

(1) Orient the edges of G arbitrarily.

(2) Let $i : m \times 1$ vector of current going across edges.

(3) Let $v : n \times 1$ vector of voltages at each node.

(4) Let $i_{\text{ext}} : n \times 1$ denote the current being forced into and forced out of each node.

Realize that

current in = current out

$$i_{\text{ext}} = B^T i,$$

and also Ohm's Law

$$I_m i = Bv.$$

where on the left hand side we have I_m denoting the identity on m entries and i being used with reference to $v = ir$.

Using both laws, we see that

$$i_{\text{ext}} = B^T Bv = Lv.$$

So, we take i_{ext} and set it such that

$$i_{\text{ext}} x_i - x_j$$

and measure $v_i - v_j = R_e$ for edge $e = (i,j)$.

We can use x_i's to pick out the ith entry such that

$$V = L^\dagger (x_i - x_j).$$

We then get that

$$R_e = (x_i - x_j)^T L^\dagger (x_i - x_j) = (BL^\dagger B^T)_{(e,e)} = \Pi_{e,e},$$

where Π is our matrix with effective resistances on the diagonal.

Cut Sparsification Given G, we want H such that $\forall S \subseteq V$,

$$\frac{\left|f_G(S) - f_H(S)\right|}{f_G(S)} \leq \epsilon.$$

This is the same as: $\forall x \in \{0,1\}^n$, and $\forall S \subseteq V$,

$$\frac{\left|x_S^T L_G x_S - x_S^T L_H x_S\right|}{x_S^T L_G x_S} \leq \epsilon.$$

Spectral Sparsification ($\implies$ Cut Sparsification)

$$\forall x \in \mathbb{R}^n,\ \forall S \subseteq V, \qquad \frac{\left|x_S^T L_G x_S - x_S^T L_H x_S\right|}{x_S^T L_G x_S} \leq \epsilon.$$

Algorithm 22: SROQ

```
Let q = O((n log n)/ε²).
for i = 1:q do
    // R_e effective resistance between nodes incident to e.
    Sample edge e with probability p_e = R_e/(n−1).
    H += e adding weight 1/(q p_e).
end
```

where $\Pi \in \mathbb{R}^{m\times m}$ and

$$\Pi = BL_G^\dagger B^T, \qquad \Pi_{ee} = R_e$$
$$\text{kernel}(B) = \text{kernel}(L) = \text{span}\{\mathbf{1}\}.$$

Recall that $L^\dagger L$ is a projector onto the sub-space of eigenvectors corresponding to non-zero eigenvalues, since

$$L^\dagger = \sum_{\lambda_i \neq 0} \frac{1}{\lambda_i} v_i v_i^T.$$

Properties of Π matrix

Theorem 39. *Π is a projector, i.e. $\Pi^2 = \Pi$.*

Proof. $BL^\dagger \underbrace{B^T B}_{=L} L^\dagger B^T = BL^\dagger B^T$. □

Theorem 40. *$im(\Pi) = im(B)$.*

Proof. We first recognize that since $\Pi = BL^\dagger B^T$, then clearly whatever Π maps onto, so can B, hence im$(\Pi) \subseteq$ im(B).

We seek to show that im$(B) \subseteq$ im(Π).

There exists $x \perp$ kernel$(B) =$ span$\{\mathbf{1}\}$. Let $Bx = y$. Then,

$$\Pi y = \Pi Bx = B\underbrace{L^\dagger B^T B}_{=L} x = BL^\dagger Lx.$$

Note that $L^\dagger L$ is a projector onto span$\{\mathbf{1}_n\}^\perp$.

Hence $\Pi y = BL^\dagger Lx = Bx = y$. So $\Pi y = y$, i.e. $y \in$ im(Π). □

Theorem 41. *$tr(\Pi) = n - 1$.*

Proof. Since π a projector matrix, it's eigenvalues are 0 or 1. Further,

$$\dim(\text{im}(B)) = \dim(\text{im}(\Pi)) = \dim(n - \text{Kernel}(B)) = n - 1.$$

□

Hence for any graph, now matter how large m,

$$\text{tr}(\Pi) = \sum_{e\in E} R_e = n - 1.$$

This is why we sample each edge with probability $p_e = R_e/(n-1)$.

Now, consider the output of our algorithm. Let's look at the laplacian of the output, for a weighted and undirected H. Let

$$d_e = \#\text{ times edge } e \text{ is sampled} \times \frac{1}{qp_e}.$$

Then $E[d_e] = 1$, since we have q trials, where in each independent trial we have probability p_e of adding weight $\frac{1}{qp_e}$.

Let S denote a diagonal $m \times m$ matrix of the d_e terms,

$$S = \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_m \end{bmatrix}.$$

Then realize that

$$E[S] = I_m, \text{ hence } L_H = B^T S B$$

where the last statement follows from the definition of the Laplacian for a weighted graph.

Hence $E[L_H] = L_G$.

WE note that B could have zero entries, i.e. edges which did not get sampled at all. Hence entries of S could be zero, although they are 1 in expectation.

It remains to apply the concentration inequality. Let $\|\cdot\|_2$ denote the operator norm for matrices, i.e. the largest singular value. We have a Concentration Inequality for Rank-1 Updates.

Theorem 42. *Rudelson-Vershynin Concentration Inequality (2003). Suppose $y \in \mathbb{R}^m$ is randomly picked such that $\|y\|_2 \le M$ and $\|E[yy^T]\|_2 \le 1$, i.e. $E[y^T y] = E[\|y\|_2^2] \le 1$.*
Let q samples be drawn from Y, i.e. $y_1, y_2, \ldots, y_q$ i.i.d.
Then,

$$E\left[\|\frac{1}{q}\sum_{i=1}^{q} y_i y_i^T - E[yy^T]\|_2\right] \le Cm\sqrt{\frac{\log q}{q}},$$

for some defined constant C (whose precise value is not important to us).

In the above inequality, as $q \uparrow$, the difference goes closer to 0 in expectation. The game is now how small we can set q in order to guarantee sparsification.

Our q is the same as the q above. For us, let us now define which distribution we use to generate the y's. We use columns of Π scaled down by

$$\frac{1}{\sqrt{p_e}},$$

i.e. we draw columns of Π (q of them) and scale them each the corresponding values. With or without replacement?

Each column of Π gets drawn with probability p_e. For the column we drew from Π, after its scaled by $1/\sqrt{p_e}$, call it y. Then,

$$\|y\|_2 = \frac{1}{\sqrt{p_e}} \cdot \|\Pi(\cdot, e)\|_2$$

where since $\Pi^2 = \Pi$, we have $(\Pi^T\Pi)_{ee} =$ the column norm contained in the diagonal entry e. Hence

$$\|y\|_2 = \frac{1}{\sqrt{p_e}}\|\Pi(e,e)\|_2 = \frac{1}{\sqrt{\frac{R_e}{n-1}}} \cdot \sqrt{R_2} = \sqrt{n-1} = M, \quad (6)$$

where we actually set our M in the last equality.

Now, we check

$$E[yy^T] = \Pi\Pi^T = \underbrace{\Pi\Pi}_{(1)\text{setting}}.$$

Further, setting

$$\Pi S \Pi = \frac{1}{q}\sum_{i=1}^{q} y_i y_i^T$$

and also q large enough to get that

$$CM\sqrt{\frac{\log q}{q}} \le \frac{\epsilon}{2}.$$

We now have that

$$E\left[\|\Pi S\Pi - \Pi\Pi\|_2\right] \le \frac{\epsilon}{2},$$

which is an Absolute Error Bound on our Π matrix. We turn this into relative error below.

By Markov, with probability $\ge 1/2$,

$$\|\Pi S\Pi - \Pi\Pi\|_2 \le \epsilon.$$

By definition of $\|\cdot\|_2$ for matrices (using Rayleigh Quotient),

$$\iff \forall y \in \mathbb{R}^m, \ \frac{\left|y^T\pi(S-I)\Pi y\right|}{y^T y} \le \epsilon.$$

$$\implies \forall y \in \text{im}(B), \text{ this result holds.} \quad \text{But } \Pi y = y, \text{ hence}$$

$$\frac{\left|y^T(S-I)y\right|}{y^T y} \le \epsilon$$

$$\implies \forall y = Bx \ s.t. \ x \perp \mathbf{1} (i.e. 1^T x = 0)$$

$$\text{we have that } \frac{\left|(Bx)^T\pi(S-I)\Pi Bx\right|}{(Bx)^T(Bx)} \le \epsilon$$

$$\iff \frac{\left|x^T L_H x - x^T L_G x\right|}{x^T L_G x} \le \epsilon \quad \forall x \perp \mathbf{1}_n.$$

Here, we have a sparsification guarantee for all $x \notin \text{kernel}(L)$. Notice that if $x \in \text{kernel}(L)$, we get sparsification for free since $Bx = 0$. Hence we have sparisfication for all $x \in \mathbb{R}^n$.

Matroids and Submodularity

Overview Today is a capstone lecture. We generalize Minimum Spanning Trees and the notion of linear independence between vectors via Matroids. We also generalize Cuts via Submodularity.

Matroids Let E be a finite universe of items (a *Ground Set*). Let $I \subseteq 2^E$ be a collection of *all* subsets of E (i.e. the power-set of E). The pair (E, I) is a Matroid $\iff$
(1) If $A \in I$, then $\forall B \subseteq A$, $B \in I$. (if some subset of vectors is independent, of course a subset of these is also independent)
(2) If $A, B \in I$ and $|A| > |B|$, then $\exists v \in A$, $v \notin B$ such that $B \cup \{v\} \in I$. (we can add an independent vector not included in our original subset of independent vectors, and the resulting set still independent).

Vector-space Matroid E.g. Let E denote a set of 1-million arbitrary vectors in $\mathbb{R}^3$. Let I denote the set of linearly independent subsets. We verify that (E, I) a matroid. (1) satisfied by properties of independent vectors. For (2): the space spanned by A larger than the space spanned by B, i.e. dimspan(A) > dimspan(B) since $|A| > |B|$ and they both belong to I. Hence we can add an element to b and still retain its membership to I.

Cardinality k-matroid E.g. For any ground-set E. Let I be the set of all subsets of E with size $\le k$. This is called a "Cardinality k-matroid".

Graphic Matroid E.g. "Graphic Matroid". Let E be the set of edges of a graph $G(V, E)$. Let $I = \{$acyclic sub-graphs$\}$, i.e. the set of forests (they don't have to be connected). Then,
(1) Realize that a sub-graph of a forest is still a forest (i.e. acyclic). That is, if some set of edges has no cycle, then any subset of these edges also has no cycle.
(2) Let A, B be forests with $|A| > |B|$, i.e. A has more edges than B. A and B are edge sets such that when installed on V of G, they do not introduce any cycles.
To add an edge to a forest, without introducing a cycle, we must add an edge between two different connected components. We seek to add an edge from A to B without introducing a cycle.
We know that #CC(A) < #CC(B). To see this, realize that we start with n nodes, i.e. n-connected components. We then added edges to nodes without introducing cycles, hence we know that if $|A| > |B|$, then the number of connected components in A must be less than in B. Hence there exist u, v disconnected in B but connected in A.
Then, for all u, v that are disconnected in B but connected in A (not necessarily directly connected in A), look at the u, v path in A. Eventually, we must find an edge in the path not in B (since u, v not connected in B). Hence such an edge spans the two components A and B.

Profit Functions Define a profit function per element in the universe. E.g. for a graphic-matroid we give each edge a profit, e.g. for vectors, each vector gets a profit.
Given a matroid (E, I) and a profit function $P(E)$, we wish to solve

$$\max_{S \in I} \sum_{e \in S} P(e) \qquad (\text{ s.t. } |S| \ge k.)$$

where I denotes our set of feasible points.
As an example, MST would have $P(e) = -1 \cdot$ edge weight. Setting $k = n - 1$, we get the MST.

Algorithm 23: Maximizing Profits via Matroids

```
Order the edges according to profit, i.e.
    P(e_1) ≥ P(e_2) ≥ ... ≥ P(e_m).
S_0 ← {}.
k ← 0.
for j = 1, 2, ..., m do
    if S_k + e_j ∈ I then
        k+ = 1.
        S_k = S_{k-1} + e_j.
    end
end
Output S_0, S_1, ..., S_k.
```

This algorithm is reminiscent of Kruskal's MST algorithm, in which we greedily search through our edges (in ascending order of weight, since we want the minimum weight spanning tree) and, if it's possible to add an edge without introducing a cycle, we do so.

Theorem 43. *S_k is the largest profit independent-set of size k.*

Proof. Suppose toward contradiction that S_k not the largest profit independent-set of size k. I.e. there exists $T \in I$, where $T = \{t_1, t_2, \ldots, t_k\}$, which is more profitable than S_k.
Without loss of generality, order elements of T such that

$$p(t_1) \ge p(t_2) \ge \ldots \ge p(t_k).$$

Again WLOG, we order our set $S = \{s_1, s_2, \ldots, s_k\} \in I$ such that

$$p(s_1) \ge p(s_n) \ge \ldots \ge p(s_k).$$

We examine the *first* index p such that $p(t_p) > p(s_p)$. Such a p must exist since if not, T would not be more profitable than S. Now, let

$$A = \{t_1, t_2, \ldots, t_p\},$$
$$B = \{s_1, s_2, \ldots, s_{p-1}\},$$

so that $|A| > |B|$.
Hence by property (2) of Matroids, there exists $t_i \in A$, where $t_i \notin B$ such that $B \cup \{t_i\} \in I$. We further know that $p(t_i) \ge p(t_p)$ by the ordering above, since $i \le p$.
Further, $p(t_p) > p(s_p)$, since we looked at the *first* index such that $p(t_p) > p(s_p)$. Hence,

$$p(t_i) \underbrace{\ge}_{i \le p} p(t_p) \underbrace{\ge}_{\text{first index p}} p(s_p).$$

Hence at some point before when constructing B, we must have considered t_i for inclusion in B. In such case, we would have added t_i to set B. But then we get a contradiction, since. □

Submodularity Suppose we have two sets, A and B. We have familiar operations, such as

$$A \cap B, A \cup B,$$
$$|A|, |B|,$$
$$|A \cap B|, |A \cup B|.$$

We know that $|A \cap B| + |A \cup B| = |A| + |B|$.
Submodular Function: Given a ground set E, where $f : 2^E \to \mathbb{R}$. If

$$f(A \cap B) + f(A \cup B) \le f(A) + f(B),$$

then f is *sub-modular*.
E.g. let $f(S) = |S|$. Then $f(A \cap B) + f(A \cup B) = f(A) + f(B)$. IF this holds with equality, the function said to be *modular*.
Symmetry: implies $f(S) = f(E \backslash S)$, e.g. cut-size.
Monotonicity: If $A \subseteq B$, then $f(A) \subseteq f(B)$.
Note that the cut function is *not* monotone. This can be seen since adding nodes to a cut doesn't always increase the cut size.

	No Constraints	Symmetric Constr.	Monotone &Matroid
Min	Poly-time	Poly-time	Poly-time
Max	Generalization of max-cut. . 1/2 Approx	1/2 Approx via greedy alg.	$(1 - \frac{1}{e})$ approx.

E.g. Coverage Functions are submodular. If your ground set made up of some profits. The universe is a set of sets, $\{V_1, V_2, \ldots, V_n\}$ and $f(S) = |\cup_{v \in S} V|$. This is a coverage function, since each vector $v \in S$ covers some proportion of the population.
For example, E may be a set of celebrities and we want to pay them to advertise for us. v_i's are how much reach each celebrity has (v_i is a set of people or fans). We wish to pick k celebrities with our budget, such that we minimize the overlap to cover the most people.