CMPUT 605 Approximation Algorithms Individual Study

Keven Qiu
Instructor: Zachary Friggstad
Fall 2025

Contents

Chapter 1

Classic Approximations

These notes are taken from Vazirani's *Approximation Algorithms*, Williamson and Shmoys' *Design of Approximation Algorithms*, and Lau, Ravi, and Singh's *Iterative Methods in Combinatorial Optimization*, as well as simplifying proofs from previous courses in approximation algorithms.

Definition: α-Approximation Algorithm

For an optimization problem, it is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor α of the value of an optimal solution.

For minimization problems, we have $\alpha > 1$ and for maximization problems, $\alpha < 1$.

1.1 Vertex Cover

Problem: Vertex Cover

Given an undirected graph $G = (V, E)$ and a cost function $c : V \to \mathbb{Q}^+$, find a min cost vertex cover.

A way to establish an approximation guarantee is by lower bounding OPT. For cardinality vertex cover, we can get a good polynomial-time computable lower bound on the size of the optimal cover.

Algorithm: 2-Approximation for Cardinality Vertex Cover

Find a maximal matching in G and output set of matched vertices.

Theorem (Cardinality Vertex Cover)

The algorithm is a 2-approximation algorithm for the cardinality vertex cover problem.

Proof. No edge can be left uncovered by the set of vertices picked. Otherwise, such an edge can have been added to the matching, contradicting maximality. Let M be this maximal matching. Since any vertex cover has to pick at least one endpoint of each matched edge, $|M| \leq \text{OPT}$. Our cover picked has cardinality $2\,|M| \leq 2 \cdot \text{OPT}$. ∎

Tight example: Complete bipartite graphs $K_{n,n}$. The algorithm will pick all $2n$ vertices, whereas optimal cover is picking one bipartition of n vertices.

The lower bound, of size of a maximal matching, is half the size of an optimal vertex cover. Consider complete graph K_n where n is odd. Then the size of any maximal matching is $\frac{n-1}{2}$, where as size of an optimal cover is $n-1$.

A NO certificate for maximum matchings in general graphs are odd set covers. These are a collection of disjoint odd cardinality subsets of V, $S_1, \ldots, S_k$ and vertices $v_1, \ldots, v_\ell$ such that each edge of G is incident with v_i or has both ends in S_j. Let C be the odd set cover, then it has cost

$$w(C) = \ell + \sum_{i=1}^{k} \frac{|S_i| - 1}{2}$$

Theorem (Generalized König)

In any graph,

$$\max_{\text{matching } M} |M| = \min_{\text{odd set cover } C} |C|$$

Corollary

In any graph,

$$\max_{\text{matching } M} |M| \leq \min_{\text{vertex cover } U} |U| \leq 2 \cdot \left(\max_{\text{matching } M} |M| \right)$$

1.2 Set Cover

Problem: Set Cover

Given a universe U of n elements, a collection of subsets of U, $\mathcal{S} = \{S_1, \ldots, S_k\}$, and a cost function $c : \mathcal{S} \to \mathbb{Q}^+$, find a min cost subcollection of $\mathcal{S}$ that covers all elements of U.

Define f as the frequency of the most frequent element. Set cover has f and $O(\log n)$ approximations. We present an $O(\log n)$-approximation here.

When $f = 2$, this is essentially the vertex cover problem.

A way to design approximation algorithms is by greedy. This is when we pick the most cost-effective choice at a particular time. Let C be the set of elements already covered. Define cost-effectiveness of a set S to be the average cost it covers new elements

$$\frac{c(S)}{|S - C|}$$

Lemma

For all $k \in \{1, \ldots, n\}$, $\text{price}(e_k) \leq \frac{\text{OPT}}{n-k+1}$

Proof. In any iteration, the leftover sets of the optimal solution can cover the remaining elements at a cost of $\leq \text{OPT}$. Therefore, among these sets, there must be a set having cost-effectiveness of

at most $\frac{\text{OPT}}{|\overline{C}|}$, where $\overline{C} = U - C$. If this were not true, then the cost of covering the remaining elements must be $> \left|\overline{C}\right| \cdot \frac{\text{OPT}}{|\overline{C}|} > \text{OPT}$, contradicting that we can cover the remaining elements of cost $\leq \text{OPT}$.

In the iteration that e_k was covered, $\overline{C}$ contained at least $n - k + 1$ elements. Thus, if e_k was covered with the most cost-effective set, then

$$\text{price}(e_k) \leq \frac{\text{OPT}}{n - k + 1}$$

■

Theorem (Set Cover)

The greedy algorithm is an H_n-approximation algorithm, where $H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$.

Proof. The total cost is

$$\sum_{k=1}^{n} \text{price}(e_k) \leq \sum_{k=1}^{n} \frac{\text{OPT}}{n - k + 1} = \text{OPT}\left(\frac{1}{n} + \frac{1}{n-1} + \cdots + 1\right) = H_n \cdot \text{OPT}$$

■

Tight example: Let $\varepsilon > 0$ be a small constant. $U = \{e_1, \ldots, e_n\}$, $\mathcal{S} = \{S_0, \ldots, S_n\}$, $c(S_0) = 1 + \varepsilon, c(S_k) = \frac{1}{k}$ for $k = 1, \ldots, n$. The cost of OPT is $1 + \varepsilon$ by choosing S_0. But greedy chooses S_k which has cost $\frac{1}{k} < 1 + \varepsilon$ for all $k = 1, \ldots, n$. So total cost is H_n.

1.3 Steiner Tree

Problem: Steiner Tree

Given $G = (V, E)$ with cost function $c : E \to \mathbb{R}_{\geq 0}$ and $V = R \cup S$ where R is the required set and S is the Steiner set, find a min cost tree in G that contains all vertices in R and any subset of S.

Theorem (Metric Steiner Tree Reduction)

There is an approximation factor preserving reduction from the Steiner tree problem to the metric Steiner tree problem.

Proof. Transform in polynomial time an instance I of G to an instance I' of the metric Steiner tree. Let G' be the complete undirected graph on V.

We construct G' as follows: $c(u, v) =$ shortest uv-path in G and the set of terminals is the same as G.

Claim 1: Cost of OPT in $G' \leq$ cost of OPT in G.
Proof of Claim 1. For all edges u, v in G, $c_{G'}(uv) \leq c_G(uv)$. ■

Claim 2: Cost of OPT in $G \leq$ cost of OPT in G'.
Proof of Claim 2. Let T' be a Steiner tree in G'. For all $uv \in E(G')$, replace uv with the shortest

uv-path to obtain the subgraph T of G. Remove edges that create cycles in T. The cost does not increase, so $c_G(uv) \leq c_{G'}(uv)$. ■

■

Algorithm: Steiner Tree 2-Approximation

Find minimum spanning tree in induced subgraph $G[R]$.

Theorem (Steiner Tree 2-Approximation)

The minimum spanning tree on R is $\leq 2 \cdot \text{OPT}$.

Proof. Sketch: Optimal Steiner tree, double each edge, find Euler tour, shortcut vertices not in R and already seen vertices and delete heaviest edge. ■

1.4 Traveling Salesman Problem

Theorem

For any polynomial-time computable function $\alpha(n)$, TSP cannot be approximated within a factor of $\alpha(n)$, unless $\mathbf{P} = \mathbf{NP}$.

Proof. Assume for contradiction that it can be $\alpha(n)$-approximated with a polynomial-time algorithm $\mathcal{A}$. We show $\mathcal{A}$ can be used to decide Hamiltonian cycle in polynomial time, implying $\mathbf{P} = \mathbf{NP}$.

Reduce the graph G on n vertices to an edge-weighted complete graph G' such that

- if G has a Hamiltonian cycle, then cost of optimal TSP tour in G' is n, and
- if G does not have a Hamiltonian cycle, then cost of optimal TSP your in G' is $> \alpha(n) \cdot n$.

Assign a weight of 1 to edges of G and weight $\alpha(n) \cdot n$ to non-edges to get G'. Now if G has a Hamiltonian cycle, then the corresponding tour has cost n in G'. Otherwise, if G has no Hamiltonian cycle, any tour in G' uses an edge of cost $\alpha(n) \cdot n$ and has cost $> \alpha(n) \cdot n$. ■

This violates the triangle inequality, so even though metric TSP is **NP**-complete, it is no longer hard to approximate.

Algorithm: Metric TSP 2-Approximation

1. Find MST T of G.
2. Double every edge of T to get Eulerian graph.
3. Find Eulerian tour $\mathcal{T}$.
4. Shortcut tour to get tour C.

Algorithm: Metric TSP $\frac{3}{2}$-Approximation (Christofides)

1. Find MST T in G.
2. Find min-cost perfect matching M on odd degree vertices.
3. $G' = T + M$.
4. Find Eulerian tour C and shortcut.

Lemma

Let $V' \subseteq V$, $|V'|$ is even, and M is min-cost perfect matching on V'. Then

$$\text{cost}(M) \leq \frac{\text{OPT}}{2}$$

Proof. Take an optimal TSP tour T of G. Let T' be tour on V' by shortcutting T. By triangle inequality, $\text{cost}(T') \leq \text{cost}(T)$.

T' is the union of 2 perfect matchings on V', consisting of alternating edges of T'. Cheapest of the matchings has cost $\leq$ cost(T') /2 $\leq$ OPT /2 since M is a min-cost perfect matching. ■

Theorem (Christofides Algorithm)

Christofides is a $\frac{3}{2}$-approximation algorithm.

Proof.

$$c(C) \leq c(T) + c(M) \leq \text{OPT} + \frac{\text{OPT}}{2} = \frac{3}{2}\,\text{OPT}$$

■

1.5 Multiway Cut and k-Cuts

Problem: Multiway Cut

Given a set of terminals $S = \{s_1, \ldots, s_k\}$, find a min-cost set of edges that when removed, disconnects S.

Algorithm: Multiway Cut $\left(2 - \frac{2}{k}\right)$-Approximation

1. For each $i = 1, \ldots, k$, compute min-weight isolating cut for s_i, say C_i.
2. Discard heaviest cut C_j and output the union of all $\bigcup_{i=1}^{k} C_i \setminus C_j$.

Problem: Min k-Cut

Find min-cost set of edges whose removal leaves k connected components.

Algorithm: k-Cut $\left(2 - \frac{2}{k}\right)$-Approximation

1. Compute a Gomory-Hu tree T for G.
2. Output union C of the lightest $k-1$ cuts from the $n-1$ cuts associated with edges of T.

1.6 k-Center

Problem: k-Center

Given an undirected graph $G = (V, E)$ with distance $d_{ij} \geq 0$ for all pairs $i, j \in V$ and an integer k, find a set $S \subseteq V, |S| = k$ of k cluster centers, where we minimize the maximum distance of a vertex to its cluster center.

Algorithm: k-Center 2-Approximation

1. Pick arbitrary $i \in V$.
2. $S = \{i\}$.
3. While $|S| < k$, $S = S \cup \{\arg\max_{j \in V} d(j, S)\}$.

Theorem

The algorithm is a 2-approximation algorithm.

Proof. Let $S^* = \{j_1, \ldots, j_k\}$ be the optimal solution with associated radius r^*. This partitions V into clusters $V_1, \ldots, V_k$ where each $j \in V$ is placed in V_i if it is closest to j_i among all in S^*. Each pair of points j and j' in the same cluster V_i are $\leq 2r^*$ apart. This is from triangle inequality; $d_{jj'} \leq d_{jj_i} + d_{j_i j'} = 2r^*$.

Let $S \subseteq V$ be points selected by the greedy algorithm. If one center in S is selected from each cluster of the optimal solution S^*, then every point in V is clearly within $2r^*$ of some point in S.

However, suppose in some iteration, the algorithm selects two points j, j' in the same cluster. The distance is at most $2r^*$. Suppose j' is selected first. Then it selects j since it was the furthest from the points already in S. Hence, all points are within a distance of at most $2r^*$ of some center already selected for S. Clearly, this remains true as the algorithm adds more centers in subsequent iterations. ■

1.7 Scheduling Jobs on Parallel Machines

Problem: Scheduling on Parallel Machines

Suppose there are n jobs, m machines, processing time p_j and no release dates. Complete all jobs as soon as possible, i.e.

$$\min \max_{j=1,\dots,n} C_j$$

or the makespan of the schedule.

Algorithm: Local Search 2-Approximation

Start with any schedule and consider job j which finishes last. Check if there exists a machine to which j can be reassigned that would cause j to finish earlier. Repeat this until the last job cannot be transferred.

Theorem (Local Search 2-Approximation)

The local search for scheduling on multiple machines is a 2-approximation algorithm.

Proof. Let $C^*_{\max}$ be the length of an optimal schedule. Since each job must be processed,

$$C^*_{\max} \geq \max_{j=1,\dots,n} p_j$$

There are in total $P = \sum p_j$ units of processing to accomplish. On average a machine will be assigned P/m units of work. At least one job must have at least that much work, so

$$C^*_{\max} \geq \sum_{j=1}^{n} p_j/m$$

Let ℓ be the last job in the final schedule of the algorithm and C_ℓ is completion time. Every machine must busy from time 0 to start of job ℓ at time $S_\ell = C_\ell - p_\ell$. Partition the schedule from time 0 to S_ℓ and S_ℓ to C_ℓ.

The latter interval has length at most $C^*_{\max}$ by first inequality.

The first interval has total work being mS_ℓ, which is no more than total work to be done P, so $S_\ell \leq \sum p_j/m$.

Combining with second inequality, $S_\ell \leq C^*_{\max}$, so in total the makespan is at most $2C^*_{\max}$.

We can refine this proof even more. $S_\ell \leq \sum p_j/m$ includes p_ℓ, but S_ℓ does not include job ℓ, so

$$S_\ell \leq \sum_{j \neq \ell} p_j/m$$

and so total length is at most

$$p_\ell + \sum_{j \neq \ell} p_j/m = \left(1 - \frac{1}{m}\right) p_\ell + \sum_{j=1}^{n} p_j/m$$

Applying two lower bounds at the start, we have $\leq \left(2 - \frac{1}{m}\right) C^*_{\max}$. ∎

To show running time, we use $C_{\min}$ and show that it cannot decrease and that we never transfer the same job twice.

Algorithm: Greedy (List Scheduling) 2-Approximation

Order jobs in a list and whenever a machine becomes idle, assign next job on that machine.

If we use this schedule with local search, it would end immediately. Consider a job ℓ that is last to complete. Each machine is busy until $C_\ell - p_\ell$, since otherwise we would have assigned job ℓ to that other machine. So no transfers are possible.

Theorem

The longest processing time rule ($p_1 \geq \cdots \geq p_n$) is a $\frac{4}{3}$-approximation algorithm.

Chapter 2

Polynomial-Time Approximation Schemes

Definition: Polynomial-Time Approximation Scheme (PTAS)

Let Π be an **NP**-hard optimization problem with objective function f_Π. $\mathcal{A}$ is a polynomial-time approximation scheme if on input (I, ε) for fixed $\varepsilon > 0$, it outputs

- $f_\Pi(I, s) \leq (1 + \varepsilon) \cdot \text{OPT}$ if Π is a minimization problem.
- $f_\Pi(I, s) \geq (1 - \varepsilon) \cdot \text{OPT}$ if Π is a maximization problem.

and its running time is bounded by a polynomial in the size of I.

Definition: Fully Polynomial-Time Approximation Scheme (FPTAS)

An approximation scheme where the running time of $\mathcal{A}$ is bounded by a polynomial in the size of instance I and $1/\varepsilon$.

2.1 Knapsack

Problem: Knapsack

Given a set $I = \{1, \ldots, n\}$ of items, with specified weight and values in $\mathbb{Z}^+$ and a knapsack capacity $B \in \mathbb{Z}^+$, find a subset of items whose total weight is bounded by B and total profit is maximized.

Definition: Pseudopolynomial-Time Algorithm

An algorithm for problem Π whose running time on instance I is bounded by a polynomial in $|I_u|$ (number of bits need to write the unary size of I).

Algorithm: Pseudopolynomial-Knapsack

Let P be most valuable object, $P = \max_{i \in I} v_i$, then nP is the upper bound on the profit of any solution. Let $S_{i,v}$ be the subset of $\{1, \ldots, i\}$ whose total value is exactly v and whose total size is minimized. Let $A(i, v)$ be the size of the set $S_{i,v}$. $A(1, v)$ is known for $\{0, \ldots, nP\}$.

$$A(i+1, p) = \begin{cases} \min\{A(i,p), w_{i+1} + A(i, p - v_{i+1})\} & \text{if } v_{i+1} \leq p \\ A(i,p) & \text{otherwise} \end{cases}$$

This dynamic programming algorithm runs in $O(n^2 P)$. The maximum profit achievable is $\max\{p : A(n,p) \leq B\}$.

Algorithm: FPTAS for Knapsack

1. Given ε and $P = \max_{i \in I} v_i$, let $K = \frac{\varepsilon P}{n}$.
2. $v'_i = \lfloor \frac{v_i}{K} \rfloor$.
3. Solve Knapsack with Dynamic Programming on new profits to get S.

Theorem

For all $\varepsilon > 0$, there is an FPTAS for Knapsack that has value $\geq (1 - \varepsilon)\,\mathrm{OPT}$.

Proof. Let S^* be the optimal solution. Note that $\mathrm{OPT} \geq P$ and $\frac{v_i}{K} - 1 \leq v'_i \leq \frac{v_i}{K}$.

The last fact gives $v'_i \leq \frac{v_i}{K} \leq \frac{P}{K} \leq \frac{n}{\varepsilon}$. Since DP solves knapsack in $O(n^2 P)$, then this FPTAS runs in $O(n^3/\varepsilon)$.

Now we bound the value of S, the set outputted by our FPTAS.

$$\begin{aligned}
\sum_{i \in S} v_i &\geq K \sum_{i \in S} v'_i \\
&\geq K \sum_{i \in S^*} v'_i && \text{(Since } S \text{ is optimal for values } v'_i\text{)} \\
&\geq K \sum_{i \in S^*} \left(\frac{v_i}{K} - 1 \right) \\
&= \sum_{i \in S^*} (v_i - K) \\
&= \sum_{i \in S^*} v_i - K\,|S^*| \\
&\geq \mathrm{OPT} - nK && (|S^*| \leq n) \\
&= \mathrm{OPT} - \varepsilon P \\
&\geq \mathrm{OPT} - \varepsilon\,\mathrm{OPT} && (\mathrm{OPT} \geq P) \\
&= (1 - \varepsilon)\,\mathrm{OPT}
\end{aligned}$$

■

2.2 Strong NP-Hardness and Existence of FPTAS

Very few of the known **NP**-hard problems admit a FPTAS.

Definition: Strongly NP-Hard

A problem Π is strongly **NP**-hard if every problem in **NP** can be polynomially reduced to Π in such a way that numbers in the reduced instance are always written in unary.

A strongly **NP**-hard problem cannot have a pseudopolynomial-time algorithm, assuming $\mathbf{P} \neq \mathbf{NP}$. Therefore, knapsack is not strongly **NP**-hard.

Theorem

Let p be a polynomial and Π be an **NP**-hard minimization problem such that the objective function f_Π is integer valued and on any instance I, $\mathrm{OPT}(I) < p(|I_u|)$. If Π admits an FPTAS, then it also admits a pseudopolynomial-time algorithm.

Proof. Suppose there is an FPTAS for Π whose running time on instance I and error parameter ε is $q(|I|, 1/\varepsilon)$, where q is a polynomial.

On instance I, set the error parameter to $\varepsilon = 1/p(|I_u|)$ and run the FPTAS. Now, the solution produced will have objective function value less than or equal to

$$(1+\varepsilon)\,\mathrm{OPT}(I) < \mathrm{OPT}(I) + \varepsilon p(|I_u|) = \mathrm{OPT}(I) + 1$$

With this error parameter, the FPTAS will be forced to produce an optimal solution. The running time will be $q(|I|, p(|I_u|))$, i.e. polynomial in $|I_u|$. Therefore, we have obtained a pseudopolynomial algorithm for Π. ■

Corollary

Let Π be an **NP**-hard optimization problem satisfying the restrictions of the theorem. If Π is strongly **NP**-hard, then Π does not admit an FPTAS, assuming $\mathbf{P} \neq \mathbf{NP}$.

Proof. If Π admits an FPTAS, then it admits a pseudopolynomial-time algorithm by theorem. But then it is not strongly **NP**-hard, assuming $\mathbf{P} \neq \mathbf{NP}$, a contradiction. ■

2.3 Bin Packing

Problem: Bin Packing

Given n items I with sizes $s_1, \ldots, s_n \in (0, 1]$, find a packing in unit-sized bins that minimizes number of bins used.

The simple 2-approximation algorithm called First-Fit is as follows: Consider items in an arbitrary order. In the ith step, it has a list of partially packed bins $B_1, \ldots, B_k$. It attempts to put the item s_i in one of these bins in order. If s_i does not fit in any of these bins, it opens a new bin B_{k+1} and puts s_i in it.

If the algorithm uses m bins, then at least $m-1$ bins are more than half full. Therefore,

$$\sum_{i=1}^{n} s_i > \frac{m-1}{2}$$

Since the sum of the item sizes is a lower bound on OPT, $m - 1 < 2\,\mathrm{OPT} \implies m \leq 2\,\mathrm{OPT}$.

Theorem

For any $\varepsilon > 0$, there is no approximation algorithm having a guarantee of $\frac{3}{2} - \varepsilon$ for the bin packing problem, unless $\mathbf{P} \neq \mathbf{NP}$.

Proof. If there were such an algorithm, then we show how to solve the **NP**-hard problem of deciding if there is a way to partition n nonnegative numbers $a_1, \ldots, a_n$ into two sets, each adding up to $\frac{1}{2}\sum_i a_i$. Clearly, the answer to this question is YES iff the n items can be packed in 2 bins of size $\frac{1}{2}\sum_i a_i$.

We can think of normalizing the Partition problem instance so that $\sum_i a_i = 2$. Since the sum is 2, the optimal bin packing requires ≥ 2 bins. If we had a $\frac{3}{2} - \varepsilon$-approximation, then we can solve this using < 3 bins, which means we can solve it optimally exactly. But if there is no solution to the Partition problem, we need ≥ 3 bins. ■

2.3.1 Asymptotic PTAS

Definition: Asymptotic PTAS (APTAS)

A family of algorithms $\{A_\varepsilon\}$ along with a constant c where is an algorithm A_ε for each $\varepsilon > 0$ such that A_ε returns a solution of value at most $(1+\varepsilon)\,\mathrm{OPT} + c$ for minimization problems.

Theorem

For any $0 < \varepsilon \leq 1$, there is an algorithm A_ε that runs in time $n^{O(1/\varepsilon^2)}$ and finds a packing using at most $(1+\varepsilon)\,\mathrm{OPT} + 1$ bins.

Idea is to ignore small items and approximately add the large items. However, we cannot scale down items like we did with knapsack and solve with DP since we may overpack bins when returning items to original size.

We instead scale items up, but we need to scale them up in a way so that the optimum value does not increase too much.

Definition: $\mathrm{SIZE}(I)$

$$\mathrm{SIZE}(I) = \sum_{i \in I} s_i$$

Lemma

Given a packing of $I_{large} = \{i \in I : s_i \geq \varepsilon/2\}$ into b bins, we can efficiently find a packing of I using $\max\{b, (1+\varepsilon)\,\mathrm{OPT} + 1\}$ bins.

Proof. Extend the packing of large items by adding small items one at a time. Create a new bin one only if none of the current bins can hold the small item.

If no new bins were created, we have b bins.

Otherwise, let b' be the total number of bins used. Since $s_i < \varepsilon/2$ for $i \in I_{small}$, we only create bins if the other bins contain total size $\geq 1 - \varepsilon/2$.

$$(b' - 1)(1 - \varepsilon/2) \leq \text{SIZE}(I) \leq \text{OPT}$$

All bins, except possibly the last bin we created will contain $\geq 1 - \varepsilon/2$. So,

$$b' \leq \frac{\text{OPT}}{1 - \varepsilon/2} + 1 \leq (1 + \varepsilon)\,\text{OPT} + 1$$

where the inequality $\frac{1}{1-\varepsilon/2} \leq 1 + \varepsilon$ holds for $0 \leq \varepsilon \leq 1$. ■

Linear Grouping: For a given value k, create a new instance I': Order $I_{large} = \{i \in I : s_i \geq \varepsilon/2\}$ in non-increasing order $s_1 \geq s_2 \geq \cdots \geq s_{n_\ell}$ where $n_\ell = |I_{large}|$. Create groups $G_1 = \{1, \ldots, k\}, G_2 = \{k+1, \ldots, 2k\}, \ldots, G_h = \{(h-1)k+1, \ldots, \}$ with sizes $\{s_1, \ldots, s_k\}$, $\{s_{k+1}, \ldots, s_{2k}\}$, $\ldots, \{s_{(h-1)k+1}, \ldots, \}$ where the last group G_h has at most k items.

The new instance I' contains items $\{k+1, \ldots, n_\ell\} = \bigcup_{i=2}^{h} G_i$, i.e. disregard first group. For an item $i \in I'$ that is in group G_a, let $s'_i = \max\{s_i : i \in G_a\}$ (round each item's size to the largest item's size of its group).

Lemma

For each $i \in I'$, $s_{i-k} \geq s'_i \geq s_i$.

We will denote $\text{OPT}(I_{large})$ as optimal solution for I_{large}, $\text{OPT}(I)$ as optimal solution for original instance I, and $\text{OPT}(I')$ as optimal solution for I'. Clearly, $\text{OPT}(I_{large}) \leq \text{OPT}(I)$.

Lemma

For instance I' with sizes s'_i obtained from I_{large} using linear grouping,

$$\text{OPT}(I') \leq \text{OPT}(I_{large}) \leq \text{OPT}(I') + k$$

Given a packing of I' into b bins, we can efficiently find a packing of I_{large} into at most $b + k$ bins.

Proof. First inequality: Consider an optimal solution for I_{large}. Pack each item $i \in I'$ in the same bin as where $i - k \in I_{large}$ is packed. Since $s'_i \leq s_{i-k}$, this produces a feasible packing for I' using at most $\text{OPT}(I_{large})$ bins.

Second inequality: Consider a packing of I' into b bins. We can pack I_{large} by packing items $1, \ldots, k$ into their own separate bins and packing each item $i \geq k+1$ into the same bin as item $i \in I'$. Since $s_i \leq s'_i$, this produces a feasible packing of I_{large} using at most $b + k$ bins. ■

We use $k = \lfloor \varepsilon \cdot \text{SIZE}(I_{large}) \rfloor$. Consider the input I', then the number of distinct piece sizes m is $m \leq \frac{n_\ell}{k}$, because we round each item in each group up, which is also $\leq h - 1$ since h was the last

group. $\text{SIZE}(I_{large}) \geq \varepsilon n_\ell/2$. Thus, for $k = \lfloor \varepsilon \cdot \text{SIZE}(I_{large}) \rfloor$, we have

$$\begin{aligned} m &= h - 1 \\ &\leq \frac{n_\ell}{k} \\ &= \frac{n_\ell}{\lfloor \varepsilon \cdot \text{SIZE}(I_{large}) \rfloor} \\ &\leq \frac{n_\ell}{\varepsilon \cdot \text{SIZE}(I_{large})/2} \qquad (*) \\ &= \frac{2n_\ell}{\varepsilon \cdot \text{SIZE}(I_{large})} \\ &= \frac{2n_\ell}{\varepsilon \cdot \varepsilon n_\ell/2} \\ &= \frac{4}{\varepsilon^2} \end{aligned}$$

$(*)$ comes from the fact that $\lfloor x \rfloor \geq x/2$. We can assume in this case $x = \varepsilon \cdot \text{SIZE}(I_{large}) \geq 1$ since otherwise there are at most $(1/\varepsilon)/(\varepsilon/2) = 2/\varepsilon^2$ large pieces and we could apply the DP algorithm to solve the input optimally without having to do linear grouping.

After linear grouping, we are left with a bin packing input where there are a constant number of distinct piece sizes and only a constant number of pieces can fit in each bin. So we can obtain an optimal packing for I' using DP (see next section on Minimum Makespan Scheduling). Say these distinct item sizes are $a_1, \ldots, a_m$. Any bin with items from I' can be identified with a tuple of nonnegative integers $(b_1, \ldots, b_m)$ where $\sum_{j=1}^m b_j a_j \leq 1$. Furthermore, since $a_j \geq \varepsilon/2$, then the number of items in this bin is at most $2/\varepsilon$, i.e. $\sum_{j=1}^m b_j \leq 2/\varepsilon$.

Let $\mathcal{C}$ be the set of all nonzero tuples $(b_1, \ldots, b_m)$ that represent a bin of size at most 1. Since $0 \leq b_j \leq 2/\varepsilon$, the number of such tuples is $\leq (2/\varepsilon + 1)^m \leq (3/\varepsilon)^{4/\varepsilon^2}$.

For any tuples $(b_1, \ldots, b_m)$, let $f(b_1, \ldots, b_m)$ be the min number of bins required to pack the set of items consisting of b_j items of size a_j. Since $|\mathcal{C}|$ is constant, then the DP algorithm for Minimum Makespan Scheduling can be used to compute $f(\bar{b}_1, \ldots, \bar{b}_m)$ in $n^{O(1/\varepsilon^2)}$, where $\bar{b}_j$ is the number of items in I' having size a_j (bins are machines and item sizes are processing times).

The packing for I' can be used to get a packing for ungrouped input, then extend with small items greedily.

Proof of Theorem. Compute an optimum packing of I' using DP in runtime $n^{O(1/\varepsilon^2)}$. By previous lemma, we can transform this to a packing of I_{large} using at most $b = \text{OPT}(I_{large}) + k = \text{OPT}(I_{large}) + \lfloor \varepsilon \cdot \text{SIZE}(I_{large}) \rfloor$ bins. Since $\text{OPT}(I) \geq \text{OPT}(I_{large}) \geq \text{SIZE}(I_{large})$, then

$$b \leq \text{OPT}(I) + \varepsilon \,\text{OPT}(I) = (1 + \varepsilon)\,\text{OPT}(I)$$

The algorithm will open $\max\{b, (1+\varepsilon)\,\text{OPT}(I) + 1\} = (1+\varepsilon)\,\text{OPT} + 1$ bins to pack the small items, where b is the number of bins used to pack large items. ■

Algorithm: Bin Packing APTAS

1. Separate I into I_{small} and I_{large} by item size threshold of $\varepsilon/2$.
2. Linear grouping with $k = \lfloor \varepsilon \cdot \text{SIZE}(I_{large}) \rfloor$ and dynamic programming.
3. First-fit on the small items into the bins filled from previous step.

2.4 Minimum Makespan Scheduling

Problem: Minimum Makespan Scheduling

Given processing times for n jobs, $J = \{p_1, \ldots, p_n\}$, and an integer m of identical machines, find an assignment of the jobs to the m identical machines so that the completion time (makespan) is minimized.

We saw a 2-approximation using local search and greedy. However, this problem is strongly **NP**-hard, and thus, does not admit a FPTAS, assuming $\mathbf{P} \neq \mathbf{NP}$.

Theorem

There is a PTAS for Minimum Makespan Scheduling.

The algorithm uses the following theorem as a subroutine.

Theorem (Oracle)

Given a value $T \geq 0$, there is an $n^{O(1/\varepsilon^2)}$ time algorithm that either

- returns a solution with makespan at most $(1+\varepsilon)\max(T, \text{OPT})$, or
- determines there is no solution with makespan $< T$.

Furthermore, if $T \geq \text{OPT}$, then the algorithm is guaranteed to find a solution with makespan at most $(1+\varepsilon)\max(T, \text{OPT})$.

Proof of PTAS for Minimum Makespan Scheduling. Assume the previous theorem holds. A binary search can be performed to find the smallest T such that a solution with makespan $\leq (1+\varepsilon)\max(T, \text{OPT})$ exists. Let $P = \sum_{j=1}^{n} p_j$. We know $0 \leq \text{OPT} \leq P$ (by scheduling all p_j on one machine) and that OPT is an integer (since all p_j are integers). So the number of calls to the algorithm in previous theorem is $O(\log P)$.

This T is $\leq$ OPT, so the makespan is at most $(1+\varepsilon)\,\text{OPT}$. The runtime of this algorithm is $O(n^{O(1/\varepsilon^2)} \cdot \log P)$ which is polynomial in the size of the input for constant values ε (at least $\log P$ bits of the input are used just to represent the processing times p_j). ■

Now we prove the second theorem. Let $J_{small} = \{j \in J : p_j \leq \varepsilon T\}$ and $J_{large} = J - J_{small}$.

Claim 1: Given a solution of makespan $(1+\varepsilon) \cdot T$ using only jobs in J_{large}, greedily placing the jobs in J_{small} as from the 2-approximation, results in makespan at most $(1+\varepsilon) \cdot max(T, OPT)$.
Proof of Claim 1. Say machine i has the highest load. If it has no jobs in J_{small} assigned to it,

then its load is at most $(1+\varepsilon)\cdot T$.

Otherwise, let $j \in J_{small}$ was added last to the schedule. The load is at most

$$p_j + \frac{1}{m}\sum_{j=1}^{n} p_j \leq \varepsilon T + \mathrm{OPT} \leq (1+\varepsilon)\max(T, \mathrm{OPT})$$

■

Proof of Oracle. If $p_j > T$, then there is no solution with makespan at most T, so assume all processing times are at most T.

We use dynamic programming to find a solution with makespan $(1+\varepsilon)\cdot T$ over J_{large} (or else determine there is no schedule $\leq T$ makespan exists).

Let b be the smallest integer such that $1/b \leq \varepsilon$. For $\varepsilon \leq 1$, we have $b \geq 2/\varepsilon$ (if $\varepsilon > 1$, we may as well just use the 2-approximation).

Define new processing times $p'_j = \left\lfloor \frac{p_j b^2}{T} \right\rfloor \cdot \frac{T}{b^2}$ (scale p_j to integer multiples of T/b^2). Then

$$p'_j \leq p_j \leq p'_j + \frac{T}{b^2}$$

and further, $p'_j = mT/b^2$ for some $m \in \{b, b+1, \ldots, b^2\}$ ($m \geq b$ since $p_j \geq T/b$ by definition of J_{large}).

For the DP algorithm define a configuration as a tuple $(a_b, a_{b+1}, \ldots, a_{b^2})$ of nonnegative integers such that

$$\sum_{i=b}^{b^2} a_i \cdot i \cdot \frac{T}{b^2} \leq T$$

a_i represents the number of jobs with running time iT/b^2. Let $\mathcal{C}(T)$ be the set of all configurations. There is a clear correspondence between configurations and assignments of jobs to a given machine with makespan (under processing times p') at most T.

The DP table is defined as follows: Given integers $n_b, \ldots, n_{b^2} \geq 0$ where n_i indicates the number of jobs with processing time iT/b^2 that need to be run, let $f(n_b, \ldots, n_{b^2})$ be the minimum number of machines required to schedule all jobs with makespan at most T. We have the DP recurrence

$$\begin{aligned} f(0,0,\ldots,0) &= 0 \\ f(n_b, n_{b+1}, \ldots, n_{b^2}) &= 1 + \min_{\{a_b,\ldots,a_{b^2} \in \mathcal{C}(T): \forall i, a_i \leq n_i\}} f(n_b - a_b, \ldots, n_{b^2} - a_{b^2}) \end{aligned}$$

For all i, $n_i \leq n$ so the number of table entries and unique configurations $(a_b, \ldots, a_{b^2})$ are both bounded by n^{b^2}. So filling the table takes at most $n^{O(b^2)}$ time.

Using this recurrence, if $\overline{n}_b, \ldots, \overline{n}_{b^2}$ is the original configuration tuple for all jobs in J_{large}, then if $f(\overline{n}_b, \ldots, \overline{n}_{b^2}) \leq k$, output YES. Otherwise, output NO. Each machine is assigned at most b jobs since $p'_j \geq T/b$ for all $j \in J_{large}$, and the p'-makespan is $\leq T$. Therefore, true makespan is

$$\leq p'\text{-makespan } + b \cdot \max_{j \in J_{large}} (p_j - p'_j) \leq T + b \cdot \frac{T}{b^2} = (1+\varepsilon)T$$

■

Chapter 3

Linear Programming Rounding

3.1 LP Duality

We want to the constraints to be written with $\geq$ for minimization LPs and $\leq$ for maximization LPs. These are the standard form and allow us to bound the objective values using a linear combination of the constraints.

This is precisely the dual LP. The dual LP gives a lower bound on the primal and the primal LP gives an upper bound on the dual, for minimization problems.

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} c_j x_j && \textbf{(Primal-LP)} \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \quad i = 1, \ldots, m \\ & x_j \geq 0, \quad j = 1, \ldots, n \end{aligned}$$

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} b_i y_i && \textbf{(Dual-LP)} \\ \text{subject to} \quad & \sum_{i=1}^{m} a_{ij} y_i \leq c_j, \quad j = 1, \ldots, n \\ & y_i \geq 0, \quad i = 1, \ldots, m \end{aligned}$$

Theorem (LP-Duality)

The primal program has finite optimum if and only if its dual has finite optimum. Moreover, if $x^* = (x_1^*, \ldots, x_n^*)$ and $y^* = (y_1^*, \ldots, y_m^*)$ are optimal solutions for the primal and dual programs, respectively, then

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*$$

LP problems are well-characterized as feasible solutions provide certificates to "Is the optimum value less than or equal to α". Thus, LP is in $\mathbf{NP} \cap \mathrm{co}\,\mathbf{NP}$.

Theorem (Weak Duality)

If x and y are feasible solutions for the primal and dual program, respectively, then

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i$$

Theorem (Complementary Slackness Conditions)

Let x and y be primal and dual feasible solutions. Then x and y are both optimal if and only if all the following conditions are satisfied:

- Primal CS conditions: For each $1 \leq j \leq n$, either $x_j = 0$ or $\sum_{i=1}^{m} a_{ij} y_i = c_j$.
- Dual CS conditions: For each $1 \leq i \leq m$, either $y_i = 0$ or $\sum_{j=1}^{n} a_{ij} x_j = b_i$.

3.2 Min-Max Relations

Consider a flow network and we want to maximize the st-flow across the network. To make it simpler, consider another arc from t to s so that we have a circulation. The primal LP is

$$\begin{aligned} \max \quad & f_{ts} \\ \text{subject to} \quad & f_{ij} \leq c_{ij}, \quad (i,j) \in E \\ & \sum_{j:(j,i)\in E} f_{ji} - \sum_{j:(i,j)\in E} f_{ij} \leq 0, \quad i \in V \\ & f_{ij} \geq 0, \quad (i,j) \in E \end{aligned}$$

This is in standard form. If the inequality holds, then it must be satisfied with equality at each node. This is because a deficit in flow balance at one node implies a surplus at some other node.

The dual LP has variables d_{ij} and p_i corresponding to the two types of inequalities in the primal.

They are distance labels and potentials on nodes.

$$\begin{aligned} \min \quad & \sum_{(i,j)\in E} c_{ij} d_{ij} \\ \text{subject to} \quad & d_{ij} - p_i + p_j \geq 0, \quad (i,j) \in E \\ & p_s - p_t \geq 1 \\ & d_{ij} \geq 0, \quad (i,j) \in E \\ & p_i \geq 0, \quad i \in V \end{aligned}$$

Most combinatorial optimization problems are integer programs. So let (d^*, p^*) be an optimal solution to the integer dual program. The solution naturally defines an st-cut $(X, \overline{X})$. All nodes in X have potential 1 and $\overline{X}$ has potential 0. All arcs going across the cut have d value 1.

If we drop integral constraints, we have an LP-relaxation of the IP. A feasible solution to the dual LP-relaxation is a fractional solution. In principle, the best fractional st-cut could have lower capacity that the best integral cut. But for this specific problem, the polyhedron defining the set of feasible solutions to the dual program has all extreme point solutions has coordinates 0 or 1. Thus, there is always an integral optimal solution.

We can also use complementary slackness conditions to get values of the variables in the optimum.

3.3 Two Fundamental Algorithm Design Techniques

Many combinatorial optimization problems can be states as integer programs. The linear relaxation provides a natural way to lower bound the cost of the optimal solution. We cannot always expect the polyhedron for **NP**-hard problems to have integer vertices. Thus, our task is to look for a near-optimal integral solution.

1. LP-Rounding: Solve the LP and then convert the fractional solution to an integral solution.
2. Primal-Dual Schema: Using the LP-relaxation of the primal, iteratively construct an integral primal solution and feasible dual solution. A feasible solution to the dual provides a lower bound on OPT.

Definition: Integrality Gap

For a minimization problem Π, let $\mathrm{OPT}_f(I)$ denote the optimal fraction solution to instance I, then the integrality gap is

$$\sup_I \frac{\mathrm{OPT}(I)}{\mathrm{OPT}_f(I)}$$

3.4 Set Cover Revisited

3.4.1 Dual Fitting

The method of dual fitting:

1. Basic algorithm is combinatorial.
2. Using LP-relaxation and its dual, the primal integral solution found by algorithm is fully paid for by the dual computed (fully paid for means that objective value of primal is at most the objective value of dual). However, the dual is infeasible.
3. For analysis, divide the dual by a suitable factor until shrunk dual is feasible. Shrunk dual is now a lower bound on OPT and the factor is the approximation guarantee.

Set cover LP:

$$\begin{aligned} \min \quad & \sum_{S \in \mathcal{S}} c(S) x_S \\ \text{subject to} \quad & \sum_{S: e \in S} x_S \geq 1, \quad e \in U \\ & x_S \in \{0,1\}, \quad S \in \mathcal{S} \end{aligned}$$

We can relax this to have $x_S \geq 0$. $x_S \leq 1$ is redundant since we want to minimize.

Set cover dual LP:

$$\begin{aligned} \max \quad & \sum_{e \in U} y_e \\ \text{subject to} \quad & \sum_{e: e \in S} y_e \leq c(S), \quad S \in \mathcal{S} \\ & y_e \geq 0, \quad e \in U \end{aligned}$$

Whenever an LP has coefficients in constraint matrix, objective function, and right hand side as all nonnegative, the min LP is called a covering LP and the maximization LP is called a packing LP.

The greedy algorithm defines dual variables price(e) for each element e. The cover picked by greedy is fully paid for by this dual solution. However, in general this dual solution is not feasible. If we shrink this by a factor of H_n, it fits into the given set cover instance, i.e. no set is overpacked. For each e, define

$$y_e = \frac{\text{price}(e)}{H_n}$$

Now y is a feasible dual solution. We show that no set is overpacked by y. Consider a set $S \in \mathcal{S}$ consisting of k elements. Number elements in order they are covered, say $e_1, \ldots, e_k$. Consider the iteration the algorithm covers e_i. S contains $\geq k - i + 1$ uncovered elements. Thus, S itself can cover e_i at an average cost of at most $\frac{c(S)}{k-i+1}$. Since the algorithm chose the most cost-effective set in this iteration, price$(e_i) \leq c(S)/(k - i + 1)$, thus,

$$y_{e_i} \leq \frac{1}{H_i} \cdot \frac{c(S)}{k - i + 1}$$

And sum over all elements

$$\sum_{i=1}^{k} y_{e_i} \leq \frac{c(S)}{H_n} \left(\frac{1}{k} + \frac{1}{k-1} + \cdots + 1 \right) = \frac{H_k}{H_n} \cdot c(S) \leq c(S)$$

Thus, the greedy algorithm is an H_n-approximation since

$$\sum_{e \in U} \text{price}(e) = H_n \sum_{e \in U} y_e \leq H_n \cdot \text{OPT}_f \leq H_n \cdot \text{OPT}$$

3.4.2 Simple LP-Rounding

One way of converting a solution to an integral solution is rounding all nonzero variables to 1. There are examples that can increase the cost by a factor of $\Omega(n)$. However, this simple algorithm does achieve the f-approximation guarantee, where f is frequency of the most frequent element.

Algorithm: Set Cover via LP-Rounding

1. Find an optimal solution to the LP-relaxation.
2. Pick all sets S which have $x_S \geq \frac{1}{f}$.

Theorem

LP-Rounding achieves an f-approximation algorithm for set cover.

Proof. Let $\mathcal{C}$ be the collection of picked sets and consider an arbitrary element e. Since e is in at most f sets, one of these sets must be picked to the extent of at least $1/f$ in the fractional cover. Thus, e is covered by $\mathcal{C}$ and $\mathcal{C}$ is a valid set cover. The rounding process increases x_S for each set $S \in \mathcal{C}$ by a factor of at most f. Therefore, the cost of $\mathcal{C}$ is at most f times the cost of the fractional cover. ■

3.4.3 Randomized Rounding

A natural idea for rounding an optimal fractional solution is to view it as probabilities, flip coins with these biases and round. We show that each element is covered with constant probability, then we repeat this process $O(\log n)$ times, picking a set if it is chosen in any of the iterations. We get a set cover with high probability by a standard coupon collector argument.

Let $x = p$ be an optimal solution to the LP. For each set $S \in \mathcal{S}$, pick S with probability p_S, the entry corresponding to S in p. Let $\mathcal{C}$ be the collection of sets picked. The expected cost of $\mathcal{C}$ is

$$E[c(\mathcal{C})] = \sum_{S \in \mathcal{S}} \Pr[S \text{ is picked}] \cdot c(S) = \sum_{S \in \mathcal{S}} p_S \cdot c(S) = \text{OPT}_f$$

Next we compute the probability that an element $a \in U$ is covered by $\mathcal{C}$. Suppose that a occurs in k sets of $\mathcal{S}$. Let the probabilities associated with these sets be $p_1, \ldots, p_k$. Since a is fractionally covered in the optimal solution, $p_1 + p_2 + \cdots + p_k \geq 1$. Under this condition, the probability that a is covered by $\mathcal{C}$ is minimized when each $p_i = 1/k$. Thus,

$$\Pr[a \text{ is covered by } \mathcal{C}] \geq 1 - \left(1 - \frac{1}{k}\right)^k \geq 1 - \frac{1}{e}$$

Hence each element is covered with constant probability. To get a complete set cover, independently pick $d \log n$ such subcollections and compute their union, say $\mathcal{C}'$, where d is a constant such that

$$\left(\frac{1}{e}\right)^{d \log n} \leq \frac{1}{4n}$$

Now,

$$\Pr[a \text{ is not covered by } \mathcal{C}'] \leq \left(\frac{1}{e}\right)^{d \log n} \leq \frac{1}{4n}$$

Summing over all elements

$$\Pr[\mathcal{C}' \text{ is not a valid set cover}] \leq n \cdot \frac{1}{4n} \leq \frac{1}{4}$$

Clearly, $E[c(\mathcal{C}')] \leq \mathrm{OPT}_f \cdot d \log n$, so we have an $O(\log n)$-approximation algorithm. Applying Markov's inequality with $t = \mathrm{OPT}_f \cdot 4d \log n$,

$$\Pr[c(\mathcal{C}') \geq \mathrm{OPT}_f \cdot 4d \log n] \leq \frac{1}{4}$$

The probability of the union of two undesirable events is $\leq \frac{1}{2}$, hence

$$\Pr[\mathcal{C}' \text{ is a valid set cover and has cost} \leq \mathrm{OPT}_f \cdot 4d \log n] \geq \frac{1}{2}$$

Observe we can verify in polynomial time whether $\mathcal{C}'$ satisfies both these conditions. If not, repeat the entire algorithm. The expected number of repetitions needed is ≤ 2.

3.5 Half-Integral Vertex Cover

Consider the vertex cover problem with nonnegative vertex costs. The IP is

$$\begin{aligned} \min \quad & \sum_{v \in V} c_v x_v \\ \text{subject to} \quad & x_u + x_v \geq 1, \quad (u,v) \in E \\ & x_v \in \{0,1\}, \quad v \in V \end{aligned}$$

Consider the LP-relaxation. Recall an extreme point solution is a feasible solution that cannot be expressed as a convex combination of two other feasible solutions. A half-integral solution is a feasible solution in which each variable is 0, 1, or $\frac{1}{2}$.

Lemma

Let x be a feasible solution to the LP-relaxation that is not half-integral. Then, x is the convex combination of two feasible solutions and is therefore not an extreme point solution for the set of inequalities.

Proof. Consider the set of vertices for which solution x does not assign half-integral values. Partition this set as follows

$$V_+ = \left\{ v : \frac{1}{2} < x_v < 1 \right\}, \quad V_- = \left\{ v : 0 < x_v < \frac{1}{2} \right\}$$

For $\varepsilon > 0$, define two solutions

$$y_v = \begin{cases} x_v + \varepsilon, & x_v \in V_+ \\ x_v - \varepsilon, & x_v \in V_- \\ x_v, & \text{otherwise} \end{cases}, \quad z_v = \begin{cases} x_v - \varepsilon, & x_v \in V_+ \\ x_v + \varepsilon, & x_v \in V_- \\ x_v, & \text{otherwise} \end{cases}$$

By assumption, $V_+ \cup V_- \neq \emptyset$ and so x is distinct from y and z. Furthermore, x is a convex combination of y and z since $x = \frac{1}{2}(y + z)$. By choosing $\varepsilon > 0$ small enough, y and z are both feasible solutions for the LP-relaxation.

Ensuring y and z are nonnegative is easy. For edge constraints, consider $x_u + x_v > 1$. By choosing ε small enough, we can ensure that y and z do not violate the constraint for such an edge. Finally, for an edge such that $x_u + x_v = 1$, there are only three possibilities: $x_u = x_v = \frac{1}{2}$, $x_u = 0, x_v = 1$, and $u \in V_+, v \in V_-$. In all three cases, for any choice of ε,

$$x_u + x_v = y_u + y_v = z_u + z_v = 1$$

The lemma follows. ■

Theorem

An extreme point solution for the LP-relaxation is half-integral.

This theorem leads to a 2-approximation by picking all vertices that are set to $\frac{1}{2}$ or 1.

3.6 Maximum Satisfiability

With the use of LP-rounding with randomization, we can obtain a $\frac{3}{4}$-approximation algorithm for maximum satisfiability. Then we can derandomize this algorithm using the method of conditional expectation.

Problem: Maximum Satisfiability (MAX-SAT)

Given a conjunctive normal form formula f on Boolean variables $x_1, \ldots, x_n$, and nonnegative weights w_C for each clause C of f, find a truth assignment to the Boolean variables that maximizes the total weight of satisfied clauses.

We let $\mathcal{C}$ represent set of clauses of f.

$$f = \bigwedge_{C \in \mathcal{C}} C$$

Each clause is a disjunction of literals.

For a positive integer k, MAX-kSAT is the restriction in which each clause has size at most k. MAX-SAT is **NP**-hard. MAX-2SAT is **NP**-hard, whereas 2SAT is in **P**.

There are two approximation algorithms, one with factor $\frac{1}{2}$ and the other $1 - \frac{1}{e}$. The first performs better if clause sizes are large and the second if clause sizes are small. We can combine the two methods to achieve a $\frac{3}{4}$-approximation.

We let random variable W be total weight of satisfied clauses. For each clause c, let random variable W_c be the weight contributed by clause C to W. Thus, $W = \sum_{C \in \mathcal{C}} W_C$ and

$$E[W_C] = w_C \cdot \Pr[C \text{ is satisfied}]$$

3.6.1 Large Clauses Approximation

Algorithm: $\frac{1}{2}$-Approximation for MAX-SAT

Assign each variable x_i either True or False, each with probability $\frac{1}{2}$.

Lemma

For each clause C with k literals,

$$\Pr[C \text{ is satisfied}] = 1 - \frac{1}{2^k}$$

Proof. Each literal in C is false with probability $\frac{1}{2}$. Since x_i are sampled independently, C is not satisfied with probability $\frac{1}{2^k}$. ■

This randomized algorithm is a $\frac{1}{2}$-approximation since each clause has $k \geq 1$ literals. Therefore, in expectation we satisfy at least half the clauses. This is tight (consider a single clause x_1).

Theorem

The expected weight of satisfied clauses is $\geq \frac{1}{2}\,\mathrm{OPT}$.

Proof.

$$E[W] = \sum_{C \in \mathcal{C}} E[W_C] \geq \frac{1}{2} \sum_{C \in \mathcal{C}} w_C \geq \frac{1}{2}\,\mathrm{OPT}$$

where $\mathrm{OPT} \leq$ total weight of clauses in $\mathcal{C}$. ■

This algorithm favours large clauses.

Derandomizing via Method of Conditional Expectation

It is possible to derandomize a randomized algorithm. We deterministically set x_i to true that will preserve the expected value of the solution. We make these decisions sequentially, i.e. set x_1, then x_2, and so on. We set x_1 in a way that will maximize the expected value of the resulting solution.

So set x_1 to true then false, then set x_1 to whichever maximizes $E[W|x_1 =?]$. We do this until all variables are set.

3.6.2 Small Clauses Approximation

Define variables $z_i \in \{0,1\}$ to be if x_i is set to true or false and $y_C \in \{0,1\}$ to be if clause C is satisfied. The LP-relaxation for MAX-SAT is

$$\begin{aligned} \max \quad & \sum_{C \in \mathcal{C}} w_C y_C \\ \text{subject to} \quad & \sum_{i: x_i \in C} z_i + \sum_{i: \overline{x}_i \in C} (1 - z_i) \geq y_C, \quad C \in \mathcal{C} \\ & 0 \leq z_i \leq 1, \quad 1 \leq i \leq n \\ & 0 \leq y_C \leq 1, \quad C \in \mathcal{C} \end{aligned}$$

The constraint for clause C ensures that y_C can be set to 1 only if at least one of the literals in C is set to true.

Note that $\mathrm{OPT} \leq \mathrm{OPT}_f$.

Algorithm: $(1-1/e)$-Approximation

Let (y^*, z^*) be an optimum LP solution. Set x_i to be True with probability z_i^* and False with probability $1 - z_i^*$.

Lemma

For any clause C with k literals,

$$\Pr[C \text{ is satisfied}] \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot y_C^*$$

Proof.

$$\begin{aligned}
\Pr[C \text{ is satisfied}] &= 1 - \Pr[C \text{ is not satisfied}] \\
&= 1 - \prod_{x_i \in C} (1 - z_i^*) \prod_{\overline{x}_i \in C} z_i^* \\
&\geq 1 - \left(\frac{\sum_{x_i \in C} (1 - z_i^*) + \sum_{\overline{x}_i \in C} z_i^*}{k} \right)^k && \text{by AM-GM Inequality} \\
&= 1 - \left(\frac{k - \sum_{x_i \in C} z_i^* - \sum_{\overline{x}_i \in C} (1 - z_i^*)}{k} \right)^k \\
&\geq 1 - \left(1 - \frac{y_C^*}{k}\right)^k
\end{aligned}$$

where the AM-GM inequality is $\sqrt[k]{\prod_i a_i} \leq \frac{1}{k} \sum_i a_i$.

Now consider the function $g(x) = 1 - \left(1 - \frac{1}{x}\right)^k$ on $[0, 1]$. This function g is concave on this interval $[0, 1]$. g is greater than the line from $(0, g(0))$ to $(1, g(1))$, i.e.

$$g(t) \geq (1 - t) \cdot g(0) + t \cdot g(1)$$

by concavity of g on $[0, 1]$. So for $t = y_C^*$,

$$\Pr[C \text{ is satisfied}] \geq g(y_C^*) \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot y_C^*$$

■

Theorem

The expected weight of satisfied clauses is $\geq \left(1 - \frac{1}{e}\right) \mathrm{OPT}_f \geq \left(1 - \frac{1}{e}\right) \mathrm{OPT}$.

Proof. Observe that $\left(1-\frac{1}{k}\right)^k \leq 1-\frac{1}{e}$ for any $k \geq 1$.

$$\begin{aligned}
E[W] &= \sum_{C \in \mathcal{C}} w_C \Pr[C \text{ is satisfied}] \\
&\geq \left(1-\left(1-\frac{1}{k}\right)^k\right) \sum_{C \in \mathcal{C}} w_C y_C^* \\
&\geq \left(1-\frac{1}{e}\right) \sum_{C \in \mathcal{C}} w_C y_C^* \\
&= \left(1-\frac{1}{e}\right) \mathrm{OPT}_f \\
&\geq \left(1-\frac{1}{e}\right) \mathrm{OPT}
\end{aligned}$$

■

3.6.3 3/4-Approximation

Algorithm: $\frac{3}{4}$-Approximation for MAX-SAT

Flip a fair coin b. If $b = 1$ run the $\frac{1}{2}$-approximation for MAX-SAT, otherwise run the $\left(1-\frac{1}{e}\right)$-approximation.

Lemma

For any clause C,

$$\Pr[C \text{ is satisfied}] \geq \frac{3}{4} y_C^*$$

Proof.

$$\begin{aligned}
\Pr[C \text{ is satisfied}] &= \Pr[b=1] \cdot \Pr[C \text{ is satisfied} | b=1] \\
&\quad + \Pr[b=0] \cdot \Pr[C \text{ is satisfied} | b=0] \\
&\geq \frac{1}{2}\left(1-\frac{1}{2^k}\right) \cdot y_C^* + \frac{1}{2} \cdot \left(1-\left(1-\frac{1}{k}\right)^k\right) \cdot y_C^* \\
&= \frac{\left(1-\frac{1}{2^k}\right) + \left(1-\left(1-\frac{1}{k}\right)^k\right)}{2} \cdot y_C^* \\
&\geq \frac{3}{4} y_C^*
\end{aligned}$$

Let $f(k) = \frac{\left(1-\frac{1}{2^k}\right)+\left(1-\left(1-\frac{1}{k}\right)^k\right)}{2}$. The last inequality comes from the value of f on values of k. Notice that $f(1) = f(2) = \frac{3}{4}$, and for $k \geq 3$, the $\frac{1}{2}$-approximation has value $\geq 7/8$ while the $(1-1/e)$-approximation has value $\geq 1-1/e$, so

$$f(k) \geq \frac{7/8 + 1 - 1/e}{2} \approx 0.753 > \frac{3}{4}$$

■

Theorem

The expected weight of clauses satisfied is $\geq \frac{3}{4}\,\mathrm{OPT}$.

Proof.

$$\begin{aligned}
E[W] &= \sum_{C \in \mathcal{C}} E[W_C] \\
&= \sum_{C \in \mathcal{C}} w_C \cdot \Pr[C \text{ is satisfied}] \\
&\geq \sum_{C \in \mathcal{C}} w_C \cdot \frac{3}{4} y_C^* \\
&= \frac{3}{4} \sum_{C \in \mathcal{C}} w_C y_C^* \\
&= \frac{3}{4}\,\mathrm{OPT}_f \\
&\geq \frac{3}{4}\,\mathrm{OPT}
\end{aligned}$$

■

3.7 Multiway Cut 1.5-Approximation

For any feasible solution F, we can compute sets C_i of vertices reachable from each s_i. For any minimal solution F, the C_i must partition V. Suppose not, let S be all vertices not reachable from any s_i. Pick some j arbitrarily and add S to C_j. Let the new solution be F' by replacing the new C_j. Then $F' \subseteq F$ since for any $i \neq j$, $\delta(C_i) \in F$ and furthermore, any edge $e \in \delta(C_j)$ has some endpoint in C_i with $i \neq j$. Thus, $e \in \delta(C_i)$ and is in F also.

This gives a new formulation for an IP. For each $v \in V$, we have k variables x_v^i where it is 1 if v is assigned to C_i and 0 otherwise. We have variables z_e^i where it is 1 if $e \in \delta(C_i)$ and 0 otherwise. e will be in two different $\delta(C_i), \delta(C_j)$, so we have the objective function as

$$\frac{1}{2} \sum_{e \in E} c_e \sum_{i=1}^{k} z_e^i$$

which will give exactly the cost of edges in the solution $F = \bigcup_{i=1}^{k} \delta(C_i)$.

s_i must be assigned to C_i, so $x_{s_i}^i = 1$ for all i. To enforce $z_e^i = 1$ for $e = (u, v) \in \delta(C_i)$, we add constraints $z_e^i \geq x_u^i - x_v^i$ and $z_e^i \geq x_v^i - x_u^i$. This enforces that $z_e^i \geq |x_u^i - x_v^i|$. Since the IP is a minimization problem and z_e^i appears with a nonnegative coefficient, at optimality we have

$z_e^i = |x_u^i - x_v^i|$. Thus, $z_e^i = 1$ if one of the endpoints of e is assigned to C_i and the other is not.

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e \sum_{i=1}^{k} z_e^i \\ \text{subject to} \quad & \sum_{i=1}^{k} x_v^i = 1, \quad v \in V \\ & z_e^i \geq x_u^i - x_v^i, \quad e = (u,v) \in E \\ & z_e^i \geq x_v^i - x_u^i, \quad e = (u,v) \in E \\ & x_{s_i}^i = 1, \quad i = 1, \ldots, k \\ & x_v^i \in \{0,1\}, \quad v \in V, i = 1, \ldots, k \end{aligned} \tag{MC-IP}$$

The LP-relaxation of this IP is closely connected to the ℓ_1 metric ($\|x - y\|_1 = \sum |x^i - y^i|$). We relax the IP with $x_u^i \geq 0$. By the first constraint, each x_u lies in the k-simplex $\Delta_k = \{x \in \mathbb{R}^k : \sum x^i = 1\}$. Each terminal s_i has $x_{s_i} = e_i$. So the LP-relaxation becomes

$$\begin{aligned} \min \quad & \sum_{e=(u,v) \in E} c_e \|x_u - x_v\|_1 \\ \text{subject to} \quad & x_{s_i} = e_i, \quad i = 1, \ldots, k \\ & x_v \in \Delta_k, \quad v \in V \end{aligned} \tag{MC-LP}$$

We design an approximation around this LP-relaxation with randomized rounding. In particular, we will take all vertices that are close to a terminal s_i and put them in C_i.

For any $r \geq 0, 1 \leq i \leq k$, let $B(s_i, r) = B(e_i, r) = \{v \in V : \frac{1}{2} \|s_i - x_v\|_1 \leq r\}$.

Algorithm: $\frac{3}{2}$-Approximation Randomized Rounding for Multiway Cut

1. Let x be an optimal solution to **MC − LP**.
2. $C_i = \emptyset$ for all $i = 1, \ldots, k$.
3. Pick $r \in (0, 1)$ uniformly at random.
4. Pick a random permutation π of $\{1, \ldots, k\}$.
5. $X = \emptyset$. (Keeps track of all currently assigned vertices)
6. For $i = 1, \ldots, k-1$,
   - $C_{\pi(i)} \leftarrow B(s_{\pi(i)}, r) - X$.
   - $X \leftarrow X \cup C_{\pi(i)}$.
7. $C_{\pi(k)} \leftarrow V - X$.
8. Return $F = \bigcup_{i=1}^{k} \delta(C_i)$.

Lemma

For each $e = (u, v)$,

$$\Pr[e \in F] \leq \frac{3}{4} \|x_u - x_v\|_1$$

We prove this later.

Theorem

The randomized algorithm is a $\frac{3}{2}$-approximation algorithm for multiway cut.

Proof. Let W be a random variable denoting value of cut and Z_e be a Bernoulli random variable which is 1 if e is in the cut, so $W = \sum_{e \in E} c_e Z_e$. Let OPT be the optimum of the LP.

Claim: For each $e = (u, v)$, $\Pr[e \in F] \leq \frac{3}{4} \|x_u - x_v\|_1$. Using this claim (proof later), we can show

$$\begin{aligned} E[W] = E\left[\sum_{e \in E} c_e Z_e\right] = \sum_{e \in E} c_e E[Z_e] &= \sum_{e \in E} c_e \cdot \Pr[e \in F] \\ &\leq \sum_{e=(u,v) \in E} c_e \cdot \frac{3}{4} \|x_u - x_v\|_1 \\ &= \frac{3}{2} \cdot \frac{1}{2} \sum_{e=(u,v) \in E} c_e \|x_u - x_v\|_1 \\ &= \frac{3}{2} \text{ OPT} \end{aligned}$$

■

Lemma 1

For any index j and any two vertices $u, v \in V$, $\left|x_u^\ell - x_v^\ell\right| \leq \frac{1}{2} \|x_u - x_v\|_1$.

Proof. Without loss of generality, assume that $x_u^j \geq x_v^j$.

$$\left|x_u^\ell - x_v^\ell\right| = x_u^\ell - x_v^\ell = \left(1 - \sum_{j \neq \ell} x_u^j\right) - \left(1 - \sum_{j \neq \ell} x_v^j\right) = \sum_{j \neq \ell} (x_v^j - x_u^j) \leq \sum_{j \neq \ell} \left|x_u^j - x_v^j\right|$$

Adding $\left|x_u^\ell - x_v^\ell\right|$ to both sides,

$$2\left|x_u^\ell - x_v^\ell\right| \leq \|x_u - x_v\|_1 \implies \left|x_u^\ell - x_v^\ell\right| \leq \frac{1}{2} \|x_u - x_v\|_1$$

■

Lemma 2

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

Proof.

$$\begin{aligned} u \in B(s_i, r) &\Leftrightarrow \frac{1}{2} \|x_{s_i} - x_u\|_1 \leq r \\ &\equiv \frac{1}{2} \sum_{j=1}^{k} \left| x_{s_i}^j - x_u^j \right| \leq r \\ &\equiv \frac{1}{2} \left(\sum_{j \neq i} x_u^j + (1 - x_u^i) \right) \leq r \\ &\equiv 1 - x_u^i \leq r \qquad \left(\because \sum_{j \neq i} x_u^j = 1 - x_u^i \right) \end{aligned}$$

■

Proof of Claim. Consider an edge $e = (u, v)$. We define two events

- S_i: We say index i settles e if i is the first index in the random permutation such that at least one of $u, v \in B(s_i, r)$.
- X_i: We say index i cuts e if exactly one of $u, v \in B(s_i, r)$.

Note that S_i depends on the random permutation while X_i is independent of the random permutation. In order for e to be in the multiway cut, there must be some index i that both settles and cuts e. If this happens, then $e \in \delta(C_i)$. Thus,

$$\Pr[e \in F] = \sum_{i=1}^{k} \Pr[S_i \wedge X_i]$$

By lemma 2,

$$\Pr[X_i] = \Pr[\min(1 - x_u^i, 1 - x_v^i) \leq r < \max(1 - x_u^i, 1 - x_v^i)] = \left| x_u^i - x_v^i \right|$$

since only if r is chosen in between x_u^i and x_v^i is when u and v are cut.

Let $\ell = \arg\min_i(1 - x_u^i, 1 - x_v^i)$. In other words, s_ℓ is the terminal closest to either u or v. We claim that index $i \neq \ell$ cannot settle edge e if ℓ comes before i in π. By lemma 1 and definition of ℓ, if at least one of $u, v \in B(s_i, r)$, then at least one of $u, v \in B(s_\ell, r)$.

Also, the probability that ℓ occurs after i in the random permutation π is $\frac{1}{2}$.

- For $i \neq \ell$,

$$\begin{aligned} \Pr[S_i \wedge X_i] &= \Pr[S_i \wedge X_i | \ell \text{ occurs after } i \text{ in } \pi] \cdot \Pr[\ell \text{ occurs after } i \text{ in } \pi] \\ &\quad + \Pr[S_i \wedge X_i | \ell \text{ occurs before } i \text{ in } \pi] \cdot \Pr[\ell \text{ occurs before } i \text{ in } \pi] \\ &\leq \Pr[X_i | \ell \text{ occurs after } i \text{ in } \pi] \cdot \frac{1}{2} + 0 \end{aligned}$$

  Since X_i is independent of π, $\Pr[X_i | \ell \text{ occurs after } i \text{ in } \pi] = \Pr[X_i]$,

$$\Pr[S_i \wedge X_i] \leq \frac{1}{2} \Pr[X_i] \cdot \frac{1}{2} = \frac{1}{2} \left| x_u^i - x_v^i \right|$$

- For $i = \ell$,

$$\Pr[S_\ell \wedge X_\ell] \leq \Pr[X_\ell] = \left|x_u^\ell - x_v^\ell\right|$$

Therefore,

$$\begin{aligned}
\Pr[e \in F] = \sum_{i=1}^{k} \Pr[S_i \wedge X_i] &\leq \left|x_u^\ell - x_v^\ell\right| + \frac{1}{2}\sum_{i\neq\ell}\left|x_u^i - x_v^i\right| \\
&= \frac{1}{2}\left|x_u^\ell - x_v^\ell\right| + \frac{1}{2}\left\|x_u - x_v\right\|_1 \\
&\leq \frac{1}{4}\left\|x_u - x_v\right\|_1 + \frac{1}{2}\left\|x_u - x_v\right\|_1 \qquad \text{(Lemma 1)} \\
&= \frac{3}{4}\left\|x_u - x_v\right\|_1
\end{aligned}$$

■

3.8 Uncapacitated Facility Location

Problem: Uncapacitated Facility Location

Given a set of clients or demands D and a set of facilities F and for each client $j \in D$ and facility $i \in F$, there is a cost of c_{ij} of assigning j to i. There is a cost f_i associated with each facility $i \in F$. The goal is to choose a subset of facilities $F' \subseteq F$ as to minimize total cost of F' and cost of assigning each client $j \in D$ to the nearest facility in F', i.e.

$$\min \sum_{i\in F'} f_i + \sum_{j\in D} \min_{i\in F'} c_{ij}$$

It is common for this problem to be metric and assignment costs c_{ij} is the distance that follow the triangle inequality:

$$c_{ij} \leq c_{i\ell} + c_{k\ell} + c_{kj}$$

3.8.1 Deterministic Rounding

We have decision variables $y_i \in \{0, 1\}$ for each facility $i \in F$ if we decide to open i or not. We also have decision variables $x_{ij} \in \{0, 1\}$ for all $i \in F$ and $j \in D$ if we assign client j to facility i.

The constraint $\sum_i x_{ij} = 1$ for all $j \in D$ is to ensure that each client j is assigned to exactly one facility. The constraint $x_{ij} \leq y_i$ is to ensure that whenever j is assigned to i, then i must be open, i.e. $y_i = 1$. We obtain the LP-relaxation by replacing the variables constraints with $x_{ij} \geq 0$ and $y_i \geq 0$ since we are minimizing.

UFL-IP

$$\begin{aligned} \min \quad & \sum_{i \in F} f_i y_i + \sum_{i \in F, j \in D} c_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{i \in F} x_{ij} = 1, \quad j \in D \\ & x_{ij} \leq y_i, \quad i \in F, j \in D \\ & y_i \geq 0, \quad i \in F \\ & x_{ij} \geq 0, \quad i \in F, j \in D \end{aligned}$$

UFL-Dual

$$\begin{aligned} \max \quad & \sum_{j \in D} v_j \\ \text{subject to} \quad & \sum_{j \in D} w_{ij} \leq f_i, \quad i \in F \\ & v_j - w_{ij} \leq c_{ij}, \quad i \in F, j \in D \\ & w_{ij} \geq 0, \quad i \in F, j \in D \end{aligned}$$

We would like to use information from the solution to the LP to find a low-cost integral solution. If a client j is fractionally assigned to a facility i, that is $x^*_{ij} > 0$, then we can consider assigning j to i.

Definition: Neighbour $N(j)$

Given an LP solution x, we say that facility i neighbours client j if $x_{ij} > 0$.

$$N(j) = \{i \in F : x_{ij} > 0\}$$

Lemma

If (x^*, y^*) is an optimal solution to the facility location LP and (v^*, w^*) is an optimal solution to its dual, then $x^*_{ij} > 0$ implies $c_{ij} \leq v^*_j$.

Proof. By complementary slackness, $x^*_{ij} > 0$ implies $v^*_j - w^*_{ij} = c_{ij}$. Since $w^*_{ij} \geq 0$, then $c_{ij} \leq v^*_j$. ■

Neighbouring facilities are useful: if we open a set of facilities S such that for all clients $j \in D$, there exists an open facility $i \in N(j)$, and then the cost of assigning j to i is no more than v^*_j by the lemma. Thus, the total assignment cost is no more than $\sum_{j \in D} v^*_j \leq \text{OPT}$.

Unfortunately, S can have high cost. Suppose we can partition some subset $F' \subseteq F$ into sets F_k such that $F_k = N(j_k)$ for some client j_k. Then if we open the cheapest facility $i_k \in N(j_k)$, we can bound the cost of i_k by

$$f_{i_k} = f_{i_k} \sum_{i \in N(j_k)} x^*_{ij_k} \leq \sum_{i \in N(j_k)} f_i x^*_{ij_k}$$

where equality comes from the first constraint of the UFL LP and the inequality comes from the choice of i_k as the cheapest facility in F_k. Using the LP constraint $x_{ij} \leq y_i$,

$$f_{i_k} \leq \sum_{i \in N(j_k)} f_i x^*_{ij_k} \leq \sum_{i \in N(j_k)} f_i y^*_i$$

Summing over all facilities that we can open,

$$\sum_k f_{i_k} \leq \sum_k \sum_{i \in N(j_k)} f_i y^*_i = \sum_{i \in F'} f_i y^*_i \leq \sum_{i \in F} f_i y^*_i$$

where the equality follows since $N(j_k)$ partitions F'. This scheme bounds the facility costs of open facilities by the facility cost of the LP solution.

Opening facilities this way does not guarantee us that every client will be a neighbour of an open facility. However, we can take advantage of the fact that assignment costs obey triangle inequality and make sure that clients are not too far from an open facility.

Definition: $N^2(j)$

$N^2(j)$ is the set of all neighboring clients of the neighboring facilities of client j.

$$N^2(j) = \{k \in D : \text{client } k \text{ neighbors some facility } i \in N(j)\}$$

Our algorithm loops until all clients are assigned to some facility. In each loop, it picks the client j_k that minimizes v_j^*. It then opens the cheapest facility i_k in the neighborhood of $N(j_k)$ and assigns j_k and all previously unassigned clients in $N^2(j_k)$ to i_k.

Algorithm: 4-Approximation Deterministic LP Rounding

1. Solve LP to get optimal primal solution (x^*, y^*) and dual solution (v^*, w^*).
2. $C \leftarrow D, k \leftarrow 0$.
3. While $C \neq \emptyset$,
   - $k \leftarrow k + 1$.
   - Choose $j_k \in C$ that minimizes v_j^* over all $j \in C$.
   - Choose $i_k \in N(j_k)$ to be the cheapest facility in $N(j_k)$.
   - Assign j_k and all unassigned clients in $N^2(j_k)$ to i_k.
   - $C \leftarrow C - \{j_k\} - N^2(j_k)$.

Theorem

The algorithm is a 4-approximation algorithm for the Uncapacitated Facility Location problem.

Proof. We have shown that $\sum_k f_{i_k} \leq \sum_{i \in F} f_i y_i^* \leq \text{OPT}$. Fix an iteration k and let $j = j_k$ and $i = i_k$. By the lemma, the cost of assigning j to i is $c_{ij} \leq v_j^*$.

Consider the cost of an unassigned client $\ell \in N^2(j)$ to facility i, where client ℓ neighbors facility h that neighbors client j, then apply triangle inequality and the lemma,

$$c_{i\ell} \leq c_{ij} + c_{hj} + c_{h\ell} \leq v_j^* + v_j^* + v_\ell^*$$

Recall we selected client j in this iteration because among all currently unassigned clients, it has the smallest dual variable v_j^*. However, ℓ is still unassigned, so its dual must be at least that of v_j^*, i.e. $v_j^* \leq v_\ell^*$. So, we conclude $c_{i\ell} \leq 3v_\ell^*$.

Thus, the solution constructed has facility cost at most OPT and assignment cost at most $3 \sum_{j \in D} v_j^* \leq 3\,\text{OPT}$ (by weak duality), for a total of $4\,\text{OPT}$. ■

Theorem

There is no α-approximation algorithm for the metric uncapacitated facility location problem with constant $\alpha < 1.463$ unless each problem in **NP** has an $O(n^{O(\log \log n)})$ time algorithm.

3.8.2 Randomized Rounding

The 4-approximation deterministic rounding algorithm works by choosing an unassigned client j that minimizes the value of v_j^* among all remaining unassigned clients, opening the cheapest facility in $N(j)$, then assigning j and all clients in $N^2(j)$ to this facility. This gave us a solution of cost

$$\sum_{i \in F} f_i y_i^* + 3 \sum_{j \in D} v_j^* \leq 4\,\mathrm{OPT}$$

We bound $\sum_{i \in F} f_i y_i^* \leq \mathrm{OPT}$, whereas we can have a stronger bound

$$\sum_{i \in F} f_i y_i^* + \sum_{i \in F, j \in D} c_{ij} x_{ij}^* \leq \mathrm{OPT}$$

The idea is that once we select a client j, instead of opening the cheapest facility in $N(j)$, we open facility $i \in N(j)$ with probability x_{ij}^* (since $\sum_{i \in N(j)} x_{ij}^* = 1$). This can amortize the costs over all possible choices of facilities in $N(j)$.

Define $C_j^* = \sum_{i \in F} c_{ij} x_{ij}^*$, that is, the assignment cost incurred by client j in the LP solution (x^*, y^*). We choose the unassigned client that minimizes $v_j^* + C_j^*$ over all unassigned clients in each iteration.

Algorithm: 3-Approximation Randomized LP Rounding

1. Solve LP to get optimal primal solution (x^*, y^*) and dual solution (v^*, w^*).
2. $C \leftarrow D, k \leftarrow 0$.
3. While $C \neq \emptyset$,
   - $k \leftarrow k + 1$.
   - Choose $j_k \in C$ that minimizes $v_j^* + C_j^*$ over all $j \in C$.
   - Choose $i_k \in N(j_k)$ according to the probability distribution $x_{ij_k}^*$.
   - Assign j_k and all unassigned clients in $N^2(j_k)$ to i_k.
   - $C \leftarrow C - \{j_k\} - N^2(j_k)$.

Theorem

The randomized rounding is a 3-approximation algorithm for the Uncapacitated Facility Location problem.

Proof. In iteration k, the expected cost of the facility opened is

$$\sum_{i \in N(j_k)} f_i x_{ij_k}^* \leq \sum_{i \in N(j_k)} f_i y_i^*$$

using the LP constraint $x_{ij_k} \leq y_i$. Recall the neighborhoods $N(j_k)$ form a partition of a subset of facilities so the overall expected cost of facilities opened is at most

$$\sum_k \sum_{i \in N(j_k)} f_i y_i^* \leq \sum_{i \in F} f_i y_i^*$$

We can now fix an iteration k and let client $j = j_k$ and facility $i = i_k$ opened. The expected cost of assigning j to i is

$$\sum_{i \in N(j)} c_{ij} x^*_{ij} = C^*_j$$

The expected cost of assigning an unassigned client $\ell \in N^2(j)$ to i, where client ℓ neighbors facility h which neighbors client j is at most

$$c_{i\ell} \leq c_{h\ell} + c_{hj} + \sum_{i \in N(j)} c_{ij} x^*_{ij} = c_{h\ell} + c_{hj} + C^*_j$$

By lemma in the deterministic rounding, $c_{h\ell} \leq v^*_\ell$ and $c_{hj} \leq v^*_j$, so the cost is $\leq v^*_\ell + v^*_j + C^*_j$. Then since we chose j to minimize $v^*_j + C^*_j$ among all unassigned clients, we know $v^*_j + C^*_j \leq v^*_\ell + C^*_\ell$. Hence the expected cost of assigning ℓ to i is $\leq v^*_\ell + v^*_j + C^*_j \leq 2v^*_\ell + C^*_\ell$. Thus, the total expected cost is no more than

$$\sum_{i \in F} f_i y^*_i + \sum_{j \in D} (2v^*_j + C^*_j) = \sum_{i \in F} f_i y^*_i + \sum_{i \in F, j \in D} c_{ij} x^*_{ij} + 2 \sum_{j \in D} v^*_j \leq 3\,\mathrm{OPT}$$

■

We are able to reduce the performance guarantee from 4 to 3 because of the random choice of facility allows us to include the assignment cost C^*_j in the analysis. Instead of bounding only facility cost by OPT, we bound both the facility cost and and part of the assignment cost by OPT.

3.9 Multicut

See section 5.1 for the multicut in trees problem.

Problem: Minimum Multicut

Let $G = (V, E)$ be an undirected graph with nonnegative capacity c_e for each $e \in E$. Let $\{(s_1, t_1), \ldots, (s_k, t_k)\}$ be a set of source-sink pairs, where each pair is distinct, but vertices in each pair may not be distinct. Find a set of edges whose removal separates each of the pairs of minimum capacity in G.

Note that this problem is **NP**-hard for $k = 3$.

3.9.1 General Graphs

Problem: Sum Multicommodity Flow

Let $G = (V, E)$ be an undirected graph with nonnegative capacity c_e for $e \in E$. Let $\{(s_1, t_1), \ldots, (s_k, t_k)\}$ be source-sink pairs. Maximize the sum of the commodities routed, where each commodity must satisfy flow conservation at each vertex other than its own source and sink. The sum of flow routed through an edge, in both directions, must not exceed the capacity of this edge.

For each commodity i, let P_i denote the set of all paths from s_i to t_i in G and let $\mathcal{P} = \bigcup_{i=1}^{k} P_i$. The LP will have a variable f_P for each $P \in \mathcal{P}$, which will be the flow along P.

$$\begin{aligned} \max \quad & \sum_{P \in \mathcal{P}} f_P \\ \text{subject to} \quad & \sum_{P: e \in P} f_P \leq c_e, \quad e \in E \\ & f_P \geq 0, \quad P \in \mathcal{P} \end{aligned}$$

For the dual, let d_e be the distance labels of edges.

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e d_e \\ \text{subject to} \quad & \sum_{e \in P} d_e \geq 1, \quad P \in \mathcal{P} \\ & d_e \geq 0, \quad e \in E \end{aligned} \qquad \textbf{(Multicut-LP)}$$

The dual program tries to find a distance label assignment to edges so that on each path $P \in \mathcal{P}$, the distance labels of edges add up to at least 1, or equivalently, d is feasible iff the shortest $s_i t_i$-path has length at least 1.

The tree version of this problem has LP and dual LP as a special case of the above LPs.

LP-Rounding

The dual (multicut) program can be solved in polynomial time using the ellipsoid algorithm, since there is a simple way of obtaining a separation oracle for it. Simply compute the length of a minimum $s_i t_i$-path with respect to the current distance labels. If all these lengths are ≥ 1, we have a feasible solution. Otherwise the shortest such path provides a violated inequality.

Let $F = \sum_{e \in E} c_e d_e$. Our goal is to pick a set of edges of small capacity, compared to F, that is a multicut. Let D be the set of edges with positive distance labels, i.e. $D = \{e : d_e > 0\}$. D is a multicut, but its capacity might be very large compared to F. In the optimal fractional multicut, the edges with large distance labels are more important than the small ones.

The algorithm will work on G with edge *lengths* given by d_e. The *weight* of edge e is defined to be $c_e d_e$. Let $dist(u, v)$ denote the length of the shortest path from u to v. For a set of vertices $S \subset V$, $\delta(S)$ denotes the cut $(S, \overline{S})$ and $c(S)$ is the capacity of $\delta(S)$, and $wt(S)$ denotes the weight of set S, which is roughly the sum of weights of all edges having both endpoints in S (will be defined precisely later).

The algorithm will find disjoint sets of vertices $S_1, \ldots, S_\ell, \ell \leq k$ in G called regions such that

- no region contains any source-sink pair, and for each i, either s_i or t_i is in one of the regions,
- for each region S_i, $c(S_i) \leq \varepsilon \cdot wt(S)$.

By the first condition, the union of the cuts of each region $M = \delta(S_1) \cup \cdots \cup \delta(S_\ell)$ is a multicut, and by second condition, its capacity $c(M) \leq \varepsilon \cdot F$.

Growing a region using the continuous process

The sets $S_1, \ldots, S_\ell$ are found through a region growing process. We first present the continuous process to clarify the issues. For time efficiency, the algorithm itself will use a discrete process.

Each region is found by growing a set starting from one vertex, which is the source or sink of a pair. This is called the *root* of the region. Suppose the root is s_1. We grow a ball around the root. For each radius r, define $B(s_1, r) = \{v : dist(s_1, v) \leq r\}$. Here $S(0) = \{s_1\}$ and as r increases continuously from 0, at discrete points, $B(s_1, r)$ grows adding vertices in increasing order of their distance from s_1.

Lemma

If the region growing process is terminated before the radius becomes $1/2$, then the set S that is found contains no source-sink pair.

Proof. The distance between any pair of vertices in $B(s_i, r)$ is $\leq 2r$. Since for each commodity i, $dist(s_i, t_i) \geq 1$ and the lemma follows. ■

For technical reasons, we assign weight to the root $wt(s_i) = F/k$. The weight of $B(s_i, r)$ is the sum of $wt(s_i)$ and the sum of the weights of edges or parts of edges, int he ball of radius r around s_i. Formally, for edges e having at least one endpoint in $B(s_i, r)$, let q_e denote the fraction of edge e in $B(s_i, r)$. If both endpoints of e are in $B(s_i, r)$, then $q_e = 1$. Otherwise, let $e = (u, v)$ with $u \in B(s_i, r), v \notin B(s_i, r)$, then

$$q_e = \frac{r - dist(s_i, u)}{dist(s_i, v) - dist(s_i, u)}$$

and the weight of region $B(s_i, r)$ is

$$wt(B(s_i, r)) = wt(s_i) + \sum c_e d_e q_e$$

where sum is over all edges with at least one endpoint in $B(s_i, r)$.

We want to fix ε so that we can guarantee that we will encounter the condition $c(B(s_i, r)) \leq \varepsilon \cdot wt(B(s_i, r))$ for $r < 1/2$. Observe that at each point, the rate at which the weight of the region is growing at least $c(B(s_i, r))$. Until this condition is encountered,

$$d\, wt(B(s_i, r)) \geq c(B(s_i, r))\, dr > \varepsilon \cdot wt(B(s_i, r))\, dr$$

Lemma

Picking $\varepsilon = 2\ln(k+1)$ suffices to ensure that the condition $c(B(s_i, r)) \leq \varepsilon \cdot wt(B(s_i, r))$ will be encountered before the radius becomes $1/2$.

Proof. Suppose for a contradiction, that throughout the region growing process, starting with $r = 0$ and ending at $r = 1/2$, $c(B(s_i, r)) > \varepsilon \cdot wt(B(s_i, r))$. At any point, the incremental change in the weight of the region is

$$d\, wt(B(s_i, r)) = \sum_e c_e d_e\, dq_e$$

Edges only having one endpoint in $B(s_i, r)$ will contribute to the sum. Then for $e = (u, v)$ with $u \in B(s_i, r), v \notin B(s_i, r)$,

$$c_e d_e\, dq_e = c_e \frac{d_e}{dist(s_i, v) - dist(s_i, u)}\, dr$$

Since $dist(s_1, v) \leq dist(s_1, u) + d_e$, we get $d_e \geq dist(s_i, v) - dist(s_i, u)$, and hence $c_e d_e \, dq_e \geq c_e \, dr$. This gives

$$d\, wt(B(s_i, r)) \geq c(B(s_i, r))\, dr > \varepsilon \cdot wt(B(s_i, r))\, dr$$

As long as the terminating conditions is not reached, the weight of the region increases exponentially with the radius. The initial weight of the region is F/k and the final weight is at most $F + F/k$. Integrating,

$$\int_{\frac{F}{k}}^{F+\frac{F}{k}} \frac{1}{wt(B(s_i, r))} d\, wt(B(S_i, r)) > \int_0^{1/2} \varepsilon \, dr$$

Therefore, $\ln(k+1) > \frac{1}{2}\varepsilon$. This contradicts the assumption that $\varepsilon = 2\ln(k+1)$. ■

The discrete process

The process starts with $S = \{s_1\}$ and adds vertices to S in increasing order of their distance from the root. This involves running a shortest path computation from the root. The set of vertices found by both processes are the same.

The weight of region S is now

$$wt(S) = wt(s_i) + \sum_e c_e d_e$$

where the sum is over all edges that have at least one endpoint in S and $wt(s_i) = F/k$. The discrete process stops at the first point when $c(S) \leq \varepsilon \cdot wt(S)$, where $\varepsilon = 2\ln(k+1)$. Notice that for the same set S, $wt(S)$ in the discrete process is at least as large as that in the continuous process. Therefore, the discrete process cannot terminate with a larger set than the continuous process. Hence, S contains no source-sink pair.

Finding successive regions

Find the first region with any of the sources. Successive regions are found iteratively.

Let $G_1 = G$ and S_1 be the region found in G_1. Consider a general point in the algorithm when regions $S_1, \ldots, S_{i-1}$ have been found. Now, G_i is defined to be the induced subgraph on vertices $V - (S_1 \cup \cdots \cup S_{i-1})$.

If G_i does not contain a source-sink pair, we are done. Otherwise, pick a source of a pair s_j as the root and define the weight to be F/k and grow region in G_i. All definitions of distance and weight are with respect to graph G_i. So, the terminating condition is $c_{G_i}(S_i) \leq \varepsilon \cdot wt_{G_i}(S_i)$.

In this manner, we find regions $S_1, \ldots, S_\ell, \ell \leq k$ and output the set $M = \delta_{G_1}(S_1) \cup \cdots \cup \delta_{G_\ell}(S_\ell)$. Since edges of each cut are removed from the graph for successive iterations, the sets are disjoint and $c(M) = \sum_i c_{G_i}(S_i)$. Edges with large distance labels are more likely to remain in the cut for a longer time, so more likely to be in the multicut.

Algorithm: $O(\log k)$-Approximation for Minimum Multicut

1. Find an optimal solution d to **Multicut-LP**.
2. $\varepsilon \leftarrow 2\ln(k+1), H \leftarrow G, M \leftarrow \emptyset$.
3. While $\exists (s_j, t_j)$ pair in H:
   - Grow a region S with root s_j until $c_H(S) \leq \varepsilon \cdot wt_H(S)$.
   - $M \leftarrow M \cup \delta_H(S)$.
   - $H \leftarrow H - V(S)$.
4. Output M.

Lemma

The set M found is a multicut.

Proof. We prove that no region contains a source-sink pair. In each iteration i, the sum of weights of edges of the graph and the weight defined on the current root is bounded by $F + F/k$. By proof of lemma 3.9.1, the continuous region growing process is guaranteed to encounter the terminating condition before the radius of the region becomes $1/2$. Therefore, the distance between pairs of vertices in S_i, found by the discrete process is also bounded by 1. Notice that we had defined these distances with respect to graph G_i. Since G_i is a subgraph of G, the distance between a pair of vertices in G cannot be larger than that in G_i. Hence, G_i contains no source-sink pair. ■

Lemma

$c(M) \leq 2\varepsilon \cdot F = 4\ln(k+1) \cdot F$.

Proof. In each iteration i, the terminating condition is $c_{G_i}(S_i) \leq \varepsilon \cdot wt_{G_i}(S_i)$. Since all edges contributes to the weight of at most one region. The total weight of all edges of G is F. Since each iteration helps disconnect at least one source-sink pair, the number of iterations is bounded by k. Therefore, the total weight attributed to source vertices is at most F. Thus,

$$c(M) = \sum_i c_{G_i}(S_i) \leq \varepsilon \left(\sum_i wt_{G_i}(S_i) \right) \leq \varepsilon \left(k \cdot \frac{F}{k} + \sum_e c_e d_e \right) = 2\varepsilon F = 4\ln(k+1)F$$

■

Theorem

The LP-rounding algorithm is a $O(\log k)$-approximation algorithm for minimum multicut.

Proof. The proof follows from the two lemmas above and from the fact that the value of the fractional multicut F is a lower bound on the minimum multicut. ■

Tight Example: The construction utilizes expander graphs.

Definition: Expander

A graph G in which every vertex has the same degree d and for any nonempty subset $S \subset V$, $|\delta(S)| > \min\{|S|, \left|\overline{S}\right|\}$.

Standard probabilistic arguments show almost every constant degree graph with $d \geq 3$ is an expander. Let H be such a graph with k vertices.

Source-sink pairs: Consider a BFS tree rooted at v. The number of vertices within distance $\alpha - 1$ of v is at most $1 + d + d^2 + \cdots + d^{\alpha-1} < d^\alpha$. Pick $\alpha = \lfloor \log_d k/2 \rfloor$ ensures that $\geq k/2$ vertices are at a distance $\geq \alpha$ from v.

A pair of vertices are a source-sink pair if the distance between them is at least α. There are $\Theta(k^2)$ source-sink pairs. The total capacity of edges in H is $O(k)$. Each edge in H is unit capacity. Since the distance between each source-sink pair is $\Omega(\log k)$, any flow path carrying flow uses up $\Omega(\log k)$ units of capacity. So value of maximum multicommodity flow in H is bounded by $O(k/\log k)$. We can prove that a minimum multicut M in H has capacity $\Omega(k)$, showing the integrality gap. This uses the claim that each connected component of $H - M$ at $\leq k/2$ vertices.

Chapter 4

Advanced Local Search

4.1 Uncapacitated Facility Location Problem

The local search will maintain a non-empty set of open facilities $S \subseteq F$ and an assignment σ of clients to S. The algorithm we first consider is permitting three types of changes to a solution:

- Opening an additional facility.
- Closing a facility.
- Swapping (adding and deleting).

The algorithm will always maintain that each client is assigned to its nearest open facility. We check to see if any of these changes reduces the total cost. If so, make this change, otherwise we stop and the current solution is a locally optimal solution.

Algorithm: $(3+\varepsilon)$-Approximation Local Search Algorithm for Uncapacitated Facility Location

1. Start with any set $S \subseteq F$ of open facilities and an assignment σ.
2. Open, close, or swap any facilities that make the total cost less than current S, σ.
3. Once no change can be made to make the cost better, output S, σ.

We first prove that any locally optimal solution is near-optimal. For any locally optimal solution, not doing the add operation implies the total assignment cost of the current solution is relatively small. If there are no delete or swap operations, then the total facility cost is relatively small. Together this yields an upper bound on the cost of the locally optimal solution.

Now let S^* the optimal open facilities, σ^* be the optimal assignment of clients to open facilities. Let F and F^* be the total facility cost of the current solution and the optimal one, respectively. Let C and C^* be their respective assignment costs. We know that $\text{OPT} = F^* + C^*$.

Each move consists of an update to S and an update to σ. We know that each operation will have change in cost that is non-negative. In fact, the update to the assignment σ need not be the

optimal one relative to the new choice of open facilities. Since the corresponding change in cost is greater than or equal to the change if we updated the assignment optimally, we are free to consider this suboptimal assignment update and that the overall change in cost is non-negative.

Lemma

Let S and σ be a locally optimal solution. Then

$$C \leq F^* + C^* = \text{OPT}$$

Proof. Since S is a locally optimal solution, we know that adding any facility to S does not improve the solution. In this way, we focus on a few potential changes to the current solution (we do not actually change, we just analyze them).

Consider some facility $i^* \in S^* - S$ and suppose that we open the additional facility i^* and reassign to that facility all of the clients that were assigned to i^* in the optimal solution, i.e. assign all clients j such that $\sigma^*(j) = i^*$. Since our solution is locally optimal, we know that the additional facility cost of i^* is at least as much as the improvement in cost that would result from reassigning each client optimally to its nearest facility. Hence f_{i^*} must also be more than the improvement resulting from our specific reassignment, that is

$$f_{i^*} \geq \sum_{j:\sigma^*(j)=i^*} (c_{\sigma(j)j} - c_{\sigma^*(j)j})$$

Now consider a facility i^* in both S and S^*. The same inequality holds since the local optimality of S and σ implies that each client j is currently assigned to its closest open facility and so each term in the summation on the RHS must be non-positive. Adding the inequality for each $i^* \in S^*$,

$$\sum_{i^* \in S^*} f_{i^*} \geq \sum_{i^* \in S^*} \sum_{j:\sigma^*(j)=i^*} (c_{\sigma(j)j} - c_{\sigma^*(j)j})$$

The LHS is equal to F^*. For the RHS, since each client j is assigned to exactly one facility $i^* \in S^*$ by σ^*, the double summation is the same as summing over all possible clients j. Hence, the first RHS terms sum to C and the second sum to C^*. It follows that $F^* \geq C - C^*$. ■

Now we have to show the local optimum has small facility cost. For any move that deletes a facility $i \in S$ (either delete or swap out), we must reassign all clients assigned to i.

If we are deleting, then each client must be reassigned to a facility in $S - \{i\}$. One natural way is for each client j, assign it to a facility $i^* = \sigma^*(j)$ in the fixed optimal solution. For each $i^* \in S^*$, let $\phi(i^*)$ be the facility in S closest to i^*. For each client j, if $i \neq i'$, where $i' = \phi(\sigma^*(j))$, then it seems reasonable to reassign client j to i'.

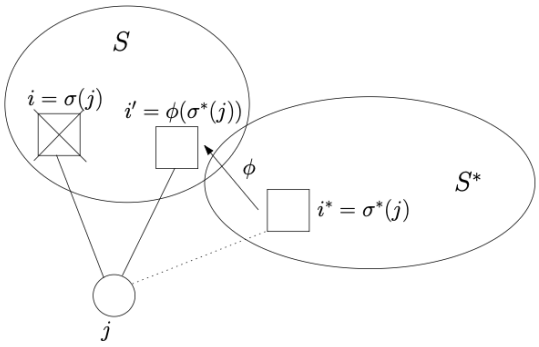
Lemma

Consider any client j for which $\sigma(j) = i$ is not equal to $i' = \phi(\sigma^*(j)) = \phi(i^*)$. Then the increase in cost of reassigning client j to i' instead of i is at most $2c_{\sigma^*(j)j}$.

Proof. Consider a client j that is currently assigned to i, where its facility in S^* being $i^* = \sigma^*(j)$, is such that i^*'s nearest facility in S, $\phi(i^*)$ is not the facility i. Let $i' = \phi(i^*)$.

By triangle inequality,

$$c_{i'j} \leq c_{i'i^*} + c_{i^*j}$$



By choice of i', we see that $c_{i'i^*} \leq c_{ii^*}$. We have

$$c_{i'j} \leq c_{ii^*} + c_{i^*j}$$

Also by triangle inequality, we have $c_{ii^*} \leq c_{ij} + c_{i^*j}$, combining

$$c_{i'j} \leq c_{ij} + 2c_{i^*j}$$

We can rewrite this as $c_{i'j} - c_{ij} \leq 2c_{i^*j} = 2c_{\sigma^*(j)j}$. ■

Lemma

Let S and σ be a locally optimal solution. Then

$$F \leq F^* + 2C^*$$

Proof. We derive an inequality based on change. Any operation must result in a non-negative change in total cost. In this construction, we give a set of moves that either deletes or swaps out every facility in S and either adds or swaps in every facility in S^*.

Suppose we want to delete a facility $i \in S$. Each client j that is current assigned to i must be reassigned to one of the remaining open facilities $S - \{i\}$. If we apply the previous lemma, we need that every client j such that $\sigma(j) = i$, we also have $\phi(\sigma^*(j)) \neq i$. We call such a facility i *safe* if for every facility $i^* \in S^*$, the facility $\phi(i^*) \in S$ closest to i^* is different from i. For any safe facility i, we can consider the local move of closing i and safely reassign its clients j to $\phi(\sigma^*(j))$, and use the bound.

We know that this local change cannot decrease the overall cost since S is locally optimal, so the savings obtained by closing the safe facility i must be no more than the increase in assignment costs incurred by reassigning all of the clients assigned to i. That is,

$$f_i \leq \sum_{j:\sigma(j)=i} 2c_{\sigma^*(j)j} \implies -f_i + \sum_{j:\sigma(j)=i} 2c_{\sigma^*(j)j} \geq 0$$

Consider a facility i that is not safe and let $R \subseteq S^*$ be the nonempty set of facilities $i^* \in S^*$ such that $\phi(i^*) = i$. Let $i' \in R$ be the one closest to i. We will derive one inequality for each facility of R based on an add move for each facility of $R - \{i'\}$, plus one swap closing the facility i and opening i'.

For the add move corresponding to $i^* \in R - \{i'\}$, we open a facility at i^* and for each client j assigned to i, we reassign it to i^* in the optimal. The change in cost cause must be non-negative,

so

$$f_{i^*} + \sum_{j:\sigma(j)=i,\sigma^*(j)=i^*} (c_{\sigma^*(j)j} - c_{\sigma(j)j}) \geq 0$$

Now for the swapping of i and opening i'. To make this swap precise, we specify a suboptimal reassignment of the clients assigned to i by σ: each client j for which $\sigma^*(j) \notin R$ is reassigned to $\phi(\sigma^*(j))$, and the rest are reassigned to i'.

The change in cost of the facilities is $f_{i'} - f_i$. There are two cases of the reassignment:

- For each client j reassigned to $\phi(\sigma^*(j))$, we are in the case governed by the previous lemma and hence the increase in assignment cost is $\leq 2c_{\sigma^*(j)j}$.
- If j is assigned to i', then the change int he assignment cost is exactly $c_{i'j} - c_{ij}$.

Combining, we obtain an upper bound on the total change in cost of the swap and we know the true change in cost is non-negative,

$$f_{i'} - f_i + \sum_{j:\sigma(j)=i,\sigma^*(j)\notin R} 2c_{\sigma^*(j)j} + \sum_{j:\sigma(j)=i,\sigma^*(j)\in R} (c_{i'j} - c_{ij}) \geq 0$$

This is if $i' \neq i$. If $i' = i$, this simply reduces to $\sum_{j:\sigma(j)=i,\sigma^*(j)\notin R} 2c_{\sigma^*(j)j} \geq 0$. For this unsafe facility i, consider the net effect of combining all of these inequalities, we get

$$-f_i + \sum_{i^*\in R} f_{i^*} + \sum_{j:\sigma(j)=i,\sigma^*(j)\notin R} 2c_{\sigma^*(j)j} + \sum_{j:\sigma(j)=i,\sigma^*(j)\in R} (c_{i'j} - c_{ij}) + \sum_{j:\sigma(j)=i,\sigma^*(j)\in R-\{i'\}} (c_{\sigma^*(j)j} - c_{\sigma(j)j}) \geq 0$$

We combine the final two summations and by showing that for each client j that appears in either summation, we can upper bound its total contribution by $2c_{\sigma^*(j)j}$.

If $\sigma^*(j) = i'$, then the contribution for client j is $c_{i'j} - c_{ij} \leq 2c_{i'j}$. If a client j has $\sigma(j) = i$ and $\sigma^*(j) \in R - \{i'\}$, its total contribution is $c_{i'j} - c_{ij} + c_{\sigma^*(j)j} - c_{\sigma(j)j} = c_{i'j} + c_{\sigma^*(j)j} - 2c_{ij}$. By triangle inequality, $c_{i'j} \leq c_{i'i} + c_{ij}$. Furthermore, by choice of i' among R, $c_{i'i} \leq c_{\sigma^*(j)i}$. By another triangle inequality, $c_{\sigma^*(j)i} \leq c_{\sigma^*(j)j} + c_{ji}$. Combining all three, we get $c_{i'j} \leq c_{\sigma^*(j)j} + 2c_{ij}$. This proves that the total contribution of j is $\leq 2c_{\sigma^*(j)j}$. Then in the inequality, we simplify to

$$-f_i + \sum_{i^*\in R} f_{i^*} + \sum_{j:\sigma(j)=i} 2c_{\sigma^*(j)j} \geq 0$$

Suppose we add the inequality for each safe facility $i \in S$ and the inequality for unsafe facility $i \in S$. Note that as we consider each of the unsafe facilities, each facility $i^* \in S^*$occurs in exactly one corresponding set R.

$$\sum_{i^*\in S^*} f_{i^*} - \sum_{i\in S} f_i + \sum_{j\in D} 2c_{\sigma^*(j)j} \geq 0$$

Thus, $F^* - F + 2C^* \geq 0$. ■

Theorem

Let S and σ be a locally optimal solution for the uncapacitated facility location problem. Then this solution has total cost that is at most $3\,\mathrm{OPT}$.

Proof. Combining the two inequalities: $C+F \le F^*+C^*+F^*+2C^* = 2F^*+3C^* \le 3(F^*+C^*) = 3\,\mathrm{OPT}$. ∎

What we really proved was a strong bound $2F^*+3C^*$. We also did not prove that the local search algorithm terminates in polynomial time, so it is not a 3-approximation. If the cost of the solution only improves by 1 with each local move, then it could take exponential time in the size of the input.

The first issue about a better bound is simple. Suppose we rescale the facility costs by dividing f_i by a factor K. If the optimal solution had assignment cost C^* and facility cost F^*, then there must now exist an assignment cost of C^* and facility cost F^*/K. Thus, the assignment cost is $\le C^* + F^*/K$ and rescaled facility cost is $\le 2C^* + F^*/K$. We obtain a solution of total cost $\le (C^* + F^*/K) + K(2C^* + F^*/K) = (1+2K)C^* + (1+1/K)F^*$.

If we set K so that the maximum of the two coefficients is as small as possible (set them equal), then $K = \sqrt{2}/2$ and the performance guarantee is $1+\sqrt{2}$.

To speed up the algorithm, rather than decreasing by any cost, we insist that the cost decreases by some factor $1-\delta < 1$. If the objective function value is initially M, and the input data is integral, then if k is chosen such that $(1-\delta)^k M < 1$, we can be sure that k iterations suffice for the algorithm to terminate. Suppose the algorithm stops whenever the current solution was nearly locally optimal, in that each possible move could not decrease the cost of the solution by a factor of $1-\delta$.

For the first lemma, in order to derive $f_{i^*} \ge \sum_{j:\sigma^*(j)=i}(c_{\sigma(j)j} - c_{\sigma^*(j)j})$, we use the fact that there are no improving moves. Now we rely on any move does not improve the solution too much.

$$f_i - \sum_{j:\sigma^*(j)=i} (c_{\sigma(j)j} - c_{\sigma^*(j)j}) \ge -\delta(C+F)$$

We add at most $|F|$ inequalities. Let $m = |F|$, then

$$F^* - C + C^* \ge -m\delta(C+F)$$

Similarly, for the other lemma, there are at most m inequalities. Each move produces a solution that is a factor of $1-\delta$ cheaper, so the RHS is $-\delta(C+F)$ instead of 0. We have

$$F^* - F + 2C^* \ge -m\delta(C+F)$$

Adding,

$$(1-2m\delta)(C+F) \le 3C^* + 2F^* \le 3\,\mathrm{OPT}$$

Hence, the bigger step local search has a performance guarantee of $\frac{3}{1-2m\delta}$.

If we set $\delta = \varepsilon/4m$, then we have a polynomial-time algorithm and a $3(1+\varepsilon)$-approximation. For the first $(1-\varepsilon/4m)^{4m/\varepsilon} \le 1/e$, so $(4m \ln M)/\varepsilon$ iterations suffice, where M could be $\sum_{i\in F} f_i + \sum_{i\in F, j\in D} c_{ij}$ by starting with all facilities open. This is a polynomial number of iterations.

The performance guarantee is $\frac{3}{1-2m\delta} = \frac{3}{1-\varepsilon/2} \le 3(1+\varepsilon)$ if $\varepsilon \le 1$. Hence, we can convert the local search into a polynomial-time approximation algorithm, losing an arbitrarily small factor in the performance guarantee.

We can combine the rescaling idea with the big steps to yield the following theorem.

Theorem

For any constant $\rho > 1 + \sqrt{2}$, the rescaled local search algorithm using bigger steps yields a ρ-approximation algorithm for the uncapacitated facility location problem.

4.2 k-Median Problem

Problem: k-Median

Consider a set of locations N in which both are potential client and facility locations. There is a cost c_{ij} of assigning j to a facility at location i. We can select a set S at most k locations at which to open a facility such that the assignment cost is minimized, i.e.

$$\sum_{j \in N} \min_{i \in S} c_{ij}$$

As with k-center, we assume the distance matrix is symmetric, satisfies triangle inequality, and has zeroes on the diagonal.

Let $S \subseteq N$ be the set of open facilities in the current solution, $S^* \subseteq N$ be the set of open facilities in the optimal solution. Let client j be assigned to its nearest open facility. Let this mapping be $\sigma(j)$ and $\sigma^*(j)$. Let C and C^* denote respectively, the total cost of the current and optimal solutions.

Algorithm: $(5 + \varepsilon)$-Approximation For k-Median

1. Let $S \subseteq N$ be any set of k locations.
2. Swap two locations $i \in S$ and $i' \in N - S$ and reassign each client to its nearest facility if i' will decrease the cost of the solution.
3. Return S when the solution is locally optimal.

Theorem

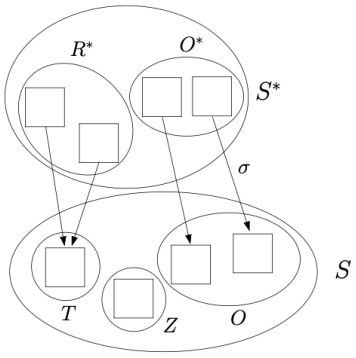
For any input to the k-median problem, any feasible solution S that is locally optimal with respect to the pairwise swap move has a cost that is at most 5 times the optimal value.

Proof. First, we construct a set of k special swaps, called the *crucial swaps*. Since the solution S is locally optimal, we know that each of these swaps does not improve the objective function of the solution. These swaps will be swapping a location $i^* \in S^*$ for a location $i \in S$. Each $i^* \in S^*$ will participate in exactly one of the k swaps, and each $i \in S$ will participate in at most 2 of these k swaps. We allow the possibility that $i^* = i$.

Observe there is a mapping from each facility $i^* \in S^*$ to a facility $\sigma(i^*) \in S$ from our current solution. This allows us to categorize facilities in S:

- $O \subseteq S$ of facilities $i \in S$ that have exactly one facility $i^* \in S^*$ with $\sigma(i^*) = i$.
- $Z \subseteq S$ of facilities $i \in S$ for which none of the facilities $i^* \in S^*$ have $\sigma(i^*) = i$.

- $T \subseteq S$ of facilities $i \in S$ such that i has at least two locations in S^* assigned to it in the current solution.



The mapping σ provides a matching between a subset $O^* \subseteq S^*$ and O. Let $R^* = S^* - O^*$ and $\ell = |R^*|$. $\ell = |Z \cup T|$ since S and S^* both have size k. Since T is the image of at least two locations in R^*, it follows $|T| \leq \ell/2$. Hence $|Z| \geq \ell/2$.

The crucial swaps: For each $i^* \in O^*$, we swap i^* with $\sigma(i^*)$. Now we build a collection of ℓ swaps, each of which swaps into the solution a distinct location in R^* and swaps out a location in Z, where each location in Z appears in at most 2 swaps. For the swaps involving locations in R^* and Z, we are free to choose any mapping provided that each element of R^* is swapped in exactly once and each element of Z is swapped out once or twice.

Consider one crucial swap, where $i^* \in S^*$ and $i \in S$ denote the swapped locations. We analyze the change in cost incurred by no longer opening a facility at location $i \in S$ and using i^* instead. Let S' denote the set of selected locations after this swap, that is $S' = S - \{i\} \cup \{i^*\}$. We also need to specify the assignment of each location in N to S'. For each j such that $\sigma^*(j) = i^*$, we assign j to i^*. For each j such that $\sigma^*(j) \neq i^*$ but $\sigma(j) = i$, we assign j to $\sigma(\sigma^*(j))$. All other j, assign to $\sigma(j)$.

We argue that $\sigma(\sigma^*(j)) \neq i$ when $\sigma^*(j) \neq i^*$. Assume for a contradiction that $\sigma(\sigma^*(j)) = i$. Then $i \in O$, since each location swapped out by a crucial swap is either in Z or O, and the former is clearly not possible by the definition of Z. Since $i \in O$, it is σ's image of exactly one element in O^*. We build a crucial swap by swapping i with that one element. Hence, $\sigma^*(j) = i^*$, but this is a contradiction.

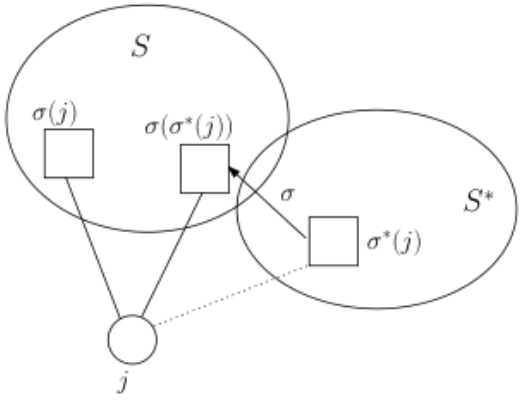
We now constructed a new set of facilities S' and an assignment of each location in N to S'. However, since there there are no improving swaps from S, any swap combined with a suboptimal assignment must not decrease overall cost.

Change of cost of swap to S': For clients j such that $\sigma^*(j) = i^*$, or both $\sigma^*(j) \neq i^*$ and $\sigma(j) = i$, the change in cost is

$$\sum_{j:\sigma^*(j)=i^*} (c_{\sigma^*(j)j} - c_{\sigma(j)j}) + \sum_{j:\sigma^*(j)\neq i^*,\sigma(j)=i} (c_{\sigma(\sigma^*(j))j} - c_{\sigma(j)j})$$

In the second summation, we can simplify it using triangle inequality shown in the figure.

$$c_{\sigma(\sigma^*(j))j} \leq c_{\sigma(\sigma^*(j))\sigma^*(j)} + c_{\sigma^*(j)j}$$



In the current solution S, $\sigma^*(j)$ is assigned to $\sigma(\sigma^*(j))$ instead of to $\sigma(j)$, so we know that

$$c_{\sigma(\sigma^*(j))\sigma^*(j)} \leq c_{\sigma(j)\sigma^*(j)}$$

By triangle inequality again,

$$c_{\sigma(j)\sigma^*(j)} \leq c_{\sigma^*(j)j} + c_{\sigma(j)j}$$

So,

$$c_{\sigma(\sigma^*(j))j} \leq c_{\sigma(j)\sigma^*(j)} + c_{\sigma^*(j)j} \leq c_{\sigma^*(j)j} + c_{\sigma(j)j} + c_{\sigma^*(j)j} = 2c_{\sigma^*(j)j} + c_{\sigma(j)j}$$

This is equivalently,

$$c_{\sigma(\sigma^*(j))j} - c_{\sigma(j)j} \leq 2c_{\sigma^*(j)j}$$

This yields a more compact upper bound on change in cost, which must be non-negative, that is, for each crucial swap i^* and i,

$$\sum_{j:\sigma^*(j)=i^*} (c_{\sigma^*(j)j} - c_{\sigma(j)j}) + \sum_{j:\sigma^*(j)\neq i^*,\sigma(j)=i} 2c_{\sigma^*(j)j} \geq 0$$

We sum this over all k crucial swaps. Recall that each $i^* \in S^*$ participates in exactly one crucial swap. Consider the first summation: we add $c_{\sigma^*(j)j}$ over all clients j which $\sigma^*(j) = i^*$ and then added for each choice of i^* in S^*. Each client j has a unique facility $\sigma^*(j) \in S^*$, so the first summation is purely summing over all $j \in N$, thus $\sum_{j\in N} c_{\sigma^*(j)j} = C^*$. The second term in the first sum is the same, so $-\sum_{j\in N} c_{\sigma(j)j} = -C$.

The second summation can be upper bounded by removing the first condition, i.e. $\sum_{j:\sigma(j)=i} 2c_{\sigma^*(j)j}$. Each facility $i \in S$ occurs in 0, 1, or 2 crucial swaps. Let n_i be the number of swaps each $i \in S$ occurs in. Thus, we can upper bound this as

$$\sum_{i\in S} \sum_{j:\sigma(j)=i} 2n_i c_{\sigma^*(j)j} \leq 4 \sum_{i\in S} \sum_{j:\sigma(j)=i} c_{\sigma^*(j)j}$$

Same as first summation, since each j is assigned to everything in S, the double sum is purely summing over all $j \in N$. We have $4\sum_{j\in N} c_{\sigma^*(j)j} = 4C^*$. Thus, substituting all the upper bounds into 4.2,

$$C^* - C + 4C^* \geq 0 \implies C \leq 5C^*$$

■

The same idea to obtain a polynomial-time algorithm from UFL can be applied. We want to make sure the swaps improve the total cost by a factor of $1 - \delta$.

Theorem

For any constant $\rho > 5$, the local search algorithm for the k-median problem that uses bigger improving swaps yields a ρ-approximation algorithm.

Proof. Let $\delta = \frac{\varepsilon}{10k}$. Since 4.2 is the change in cost for one crucial swap, we sum for all k swaps and making sure we do not decrease the cost too much, so

$$C^* - C + 4C^* \geq -k\delta C \implies (1 - k\delta)C \leq 5C^* \leq 5\,\mathrm{OPT}$$

Hence we have an approximation ratio of $\frac{5}{1-k\delta} = \frac{5}{1-\varepsilon/10} \leq 5 + \varepsilon$ for $0 < \varepsilon \leq 1$.

For the polynomial-time, we have $(1 - \varepsilon/2k)^{2k/\varepsilon} \leq 1/e$, so $(2k \ln C)/\varepsilon$ iterations suffice, where C is the cost of the our solution. ■

Chapter 5

Primal-Dual Algorithms

This method leverages complementary slackness, that is, deciding which variables in the primal to keep and adjusting dual variables until a dual constraint goes tight.

Primal Complementary Slackness: Let $\alpha \geq 1$. For each $1 \leq j \leq n$, either $x_j = 0$ or $\frac{c_j}{\alpha} \leq \sum_{i=1}^{m} a_{ij} y_i \leq c_j$.

Dual Complementary Slackness: Let $\beta \geq 1$. For each $1 \leq i \leq m$, either $y_i = 0$ or $b_i \leq \sum_{j=1}^{n} a_{ij} x_j \leq \beta \cdot b_i$.

Proposition (Primal-Dual)

If x, y are primal and dual feasible solutions satisfying the above conditions, then

$$\sum_{j=1}^{n} c_j x_j \leq \alpha\beta \sum_{i=1}^{m} b_i y_i$$

If $\alpha = \beta = 1$, then we have the original CS conditions, which ensure optimality.

5.1 Multicut and Integer Multicommodity Flow in Trees

Since G is a tree, there is a unique path between any s_i and t_i and the multicut must pick an edge on the path to disconnect them.

This problem is **NP**-hard even on trees of height 1 and unit capacity edges. Let x_e be a decision variable for if e is in the multicut or not. We write the LP-relaxation

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e && \textbf{(Tree-Multicut-LP)} \\ \text{subject to} \quad & \sum_{e \in P_i} x_e \geq 1, \quad i = \{1, \ldots, k\} \\ & x_e \geq 0, \quad e \in E \end{aligned}$$

The dual program is a multicommodity flow in G, with a separate commodity corresponding to each vertex pair (s_i, t_i). Dual variable f_i will denote the amount of this commodity is routed along the unique $s_i t_i$-path.

$$\begin{aligned} \max \quad & \sum_{i=1}^{k} f_i \\ \text{subject to} \quad & \sum_{i:e\in P_i} f_i \le c_e, \quad e \in E \\ & f_i \ge 0, \quad i \in \{1, \ldots, k\} \end{aligned}$$

Problem: Integer Multicommodity Flow

We have the same graph G and source-sink pairs, but edge capacities are all integral. Maximize the sum of the commodities routed subject to capacity constraints and routing each commodity integrally.

5.1.1 Primal-Dual Schema

We use the primal-dual method to obtain a multicut and integer multicommodity flow that are within a factor of 2 of each other for trees.

We will ensure primal complementary slackness conditions, i.e. $\alpha = 1$, and relax the dual conditions with $\beta = 2$.

Primal conditions: For each $e \in E$, $x_e \neq 0 \implies \sum_{i:e\in P_i} f_i = c_e$. Equivalently, any edge picked in the multicut must be saturated.

Relaxed dual conditions: For each $i \in \{1, \ldots, k\}$, $f_i \neq 0 \implies \sum_{e\in P_i} x_e \le 2 \cdot 1 = 2$. Equivalently, at most 2 edges can be picked from a path carrying nonzero flow.

Definition: Least Common Ancestor $\text{lca}(u, v)$

Root the tree at an arbitrary vertex. The least common ancestor of u and v is the minimum depth vertex on the path from u to v.

Algorithm: 2-Approximation for Multicut and Multicommodity Flow in Trees

1. Initialize: $f \leftarrow 0, D \leftarrow \emptyset$.
2. Flow routing: For each $v \in V$, in non-increasing order of depth, do
   - For each pair (s_i, t_i) such that $\text{lca}(s_i, t_i) = v$, greedily route integral flow from s_i to t_i.
   - Add to D all edges that were saturated in the current iteration.
3. Let $D = \{e_1, \ldots, e_\ell\}$ be the ordered list of edges.
4. Reverse delete: For $j = \ell, \ldots, 1$, if $D - \{e_j\}$ is a multicut in G, delete e_j from D.
5. Output f and D.

Lemma

Let (s_i, t_i) be a pair with nonzero flow and let $\text{lca}(s_i, t_i) = v$. At most one edge is picked in the multicut from each of the two paths s_i to v and t_i to v.

Proof. This argument will be the same for each path. Suppose two edges e and e' are picked from the $s_i v$-path, with e being the deeper edge. Clearly, e' must be in D all through reverse delete.

Consider the moment during reverse delete when edge e is being tested. Since e is not discarded, there must be a pair, say (s_j, t_j), such that e is the only edge of D on the $s_j t_j$-path. Let $u = \text{lca}(s_j, t_j)$. Since e' does not lie on the $s_j t_j$-path, u must be deeper than e', and hence deeper than v. After u has been processed, D must contain an edge from the $s_j t_j$-path, say e''.

Since nonzero flow has been routed from s_i to t_i, e must be added during or after in which v is processed. Since v is an ancestor of u, e is added after e''. So e'' must be in D when e is being tested. This contradicts the fact that at this moment e is the only edge of D on the $s_j t_j$-path. ■

Theorem

The algorithm is a 2-approximation for minimum multicut and $\frac{1}{2}$-approximation for maximum integer multicommodity flow on trees.

Proof. The flow found at the end of flow routing is maximal and since D contains all the saturated edges, D is a multicut. Since reverse delete only discards redundant edges, D is still a multicut. Thus, we have feasible solutions.

Since each edge in the multicut is saturated, the primal conditions are satisfied. By the lemma, at most two edges have been picked in the multicut from each path carrying nonzero flow. Therefore, the relaxed dual conditions are also satisfied. Hence, by the proposition, the capacity of the multicut found is within 2 times the flow. Since a feasible flow is a lower bound on the optimal multicut, a feasible multicut is an upper bound on the optimal integer multicommodity flow. ■

Corollary

On trees with integral edge capacity,

$$\max_{\text{int. flow } F} |F| \leq \min_{\text{multicut } C} c(C) \leq 2 \cdot \max_{\text{int. flow } F} |F|$$

5.2 Steiner Forest

Problem: Steiner Forest

Given an undirected graph $G = (V, E)$, a cost function $c : E \to \mathbb{Q}^+$, and a collection of disjoint subsets of V, $S_1, \ldots, S_k$, find a minimum cost subgraph in which each pair of vertices belong to the same set S_i is connected.

Define a connectivity requirement function r that maps ordered pairs of vertices to $\{0, 1\}$ as follows:

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

Now the problem is to find a minimum cost subgraph F that contains a uv-path for each pair (u, v) with $r(u, v) = 1$. In general, the solution will be a forest.

5.2.1 LP-Relaxation and Dual

Define a function on all cuts in G, $f : 2^V \to \{0, 1\}$, which specifies the minimum number of edges that must cross each cut in any feasible solution.

$$f(S) = \begin{cases} 1 & \text{if } \exists u \in S \text{ and } v \in \overline{S} \text{ such that } r(u, v) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Steiner Forest LP

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e \\ \text{subject to} \quad & \sum_{e \in \delta(S)} x_e \geq f(S), \quad S \subseteq V \\ & x_e \geq 0, \quad e \in E \end{aligned}$$

Steiner Forest Dual

$$\begin{aligned} \max \quad & \sum_{S \subseteq V} f(S) y_S \\ \text{subject to} \quad & \sum_{S : e \in \delta(S)} y_S \leq c_e, \quad e \in E \\ & y_S \geq 0, \quad S \subseteq V \end{aligned}$$

5.2.2 Primal-Dual Schema with Synchronization

Typically, in each iteration, we pick one unsatisfied complementary slackness condition, and raise the primal and dual solutions suitably. The new idea is to raise the duals in a synchronized manner. This way, it does not rectify a specific condition, but rather many possibilities simultaneously.

We say edge e *feels* dual y_S if $y_S > 0$ and $e \in \delta(S)$. The set S has been *raised* in a dual solution if $y_S > 0$. Raising either S or $\overline{S}$ has the same effect. There is no advantage in raising a set S with $f(S) = 0$ since this does not contribute to the dual objective function. An edge e is *tight* if the dual it feels equals its cost.

Let the degree of a set S be the number of picked edges crossing the cut $(S, \overline{S})$.

Primal conditions: For each $e \in E$, $x_e \neq 0 \implies \sum_{S : e \in \delta(S)} y_S = c_e$.

Relaxed dual conditions: For each $S \subseteq V, y_S \neq 0 \implies \sum_{e \in \delta(S)} x_e \leq 2 f(S)$, i.e. every raised cut has degree at most 2. However, we do not know how to ensure this condition. We can still obtain a 2-approximation by relaxing further. Raised sets will be allowed to have high degree, but we ensure that *on average*, raised duals have degree at most 2.

The algorithm starts with null primal and dual solutions. The primal solution indicates which cuts needs to be raised and the dual solution indicates which edge needs to be picked. The algorithm iteratively improves feasibility of primal and optimality of dual, until the primal is feasible.

The set S is unsatisfied if $f(S) = 1$, but there is no picked edge crossing the cut $(S, \overline{S})$. Set S is said to be active if it is inclusion-wise minimal unsatisfied set in the current iteration. Clearly, if the currently picked primal is infeasible, then there must be an unsatisfied set and an active set with respect to it.

Lemma

Set S is active if and only if it is a connected component in the currently picked forest and $f(S) = 1$.

Proof. Let S be an active set. S cannot contain part of a connected component because otherwise there will already be a picked edge in the cut $(S, \overline{S})$. Thus, S is a union of connected components. Since $f(S) = 1$, there is a vertex $u \in S$ and $v \in \overline{S}$ such that $r(u, v) = 1$. Let S' be the connected component containing u. S' is also unsatisfied and by minimality of S, $S = S'$. ■

By the characterization of active sets in the lemma, it is easy to find all active sets by raising the duals of these sets synchronously until some edge goes tight. Any one of the newly tight edges is picked and the iteration terminates. The set of edges found in the primal may contain redundant edges.

Algorithm: 2-Approximation for Steiner Forest

1. (Initialization) $F \leftarrow \emptyset$ for each $S \subseteq V$, $y_S \leftarrow 0$.
2. (Edge augmentation) While there exists an unsatisfied set, simultaneously raise y_S for each active set S, until some edge e goes tight. $F \leftarrow F \cup \{e\}$.
3. (Pruning) Return $F' = \{e \in F : F - \{e\} \text{ is primal infeasible}\}$

Lemma

At the end of the algorithm, F' and y are primal and dual feasible solutions, respectively.

Proof. At the end of step 2, F satisfies all connectivity requirements. In each iteration, dual variables of connected components only are raised. Therefore, no edge running within the same component can go tight. Thus, F is acyclic. Therefore, if $r(u, v) = 1$, there is a unique uv-path in F. Thus, each edge on this path is nonredundant and is not deleted in step 3. Hence, F' is primal feasible.

When an edge goes tight, the current iteration ends and active sets are redefined. Therefore, no edge is overtightened. Hence, y is dual feasible. ■

Lemma

Consider any iteration of the algorithm, and let C be a component with respect to the currently picked edges. If $f(C) = 0$, then $|\delta_{F'}(C)| \neq 1$.

Proof. Suppose $|\delta_{F'}(C)| = 1$ and let e be the unique edge of F' crossing the cut $(C, \overline{C})$. Since e is nonredundant (every edge in F' is nonredundant), there is a pair of vertices, say u, v, such that $r(u, v) = 1$ and e lies on the unique uv-path in F'. Since this path crosses the cut $(C, \overline{C})$ exactly once, one of these vertices must lie in C and the other in $\overline{C}$. Now, since $r(u, v) = 1$, we have $f(C) = 1$, leading to a contradiction. ■

Theorem

The algorithm achieves a 2-approximation for the Steiner Forest problem, i.e.

$$\sum_{e \in F'} c_e \leq 2 \sum_{S \subseteq V} y_S$$

Proof. We know that F' and y are primal and dual feasible solutions, respectively.

Since every picked edge is tight,

$$\sum_{e \in F'} c_e = \sum_{e \in F'} \left(\sum_{S : e \in \delta(S)} y_S \right) = \sum_{S \subseteq V} \left(\sum_{e \in \delta(S) \cap F'} y_S \right) = \sum_{S \subseteq V} |\delta_{F'}(S)| \, y_S$$

Thus, we need to show $\sum_{S \subseteq V} |\delta_{F'}(S)| \, y_S \leq 2 \sum_{S \subseteq V} y_S$.

We prove that in each iteration, the increase in the LHS is bounded by the increase in the RHS. Consider an iteration and let Δ be the extent to which active sets were raised in this iteration. Then, we show

$$\Delta \left(\sum_{S \text{ active}} |\delta_{F'}(S)| \right) \leq 2\Delta \times (\# \text{ of active sets}) \implies \frac{\sum_{S \text{ active}} |\delta_{F'}(S)|}{\# \text{ of active sets}} \leq 2$$

Notice that the number of edges crossing any active set S is due to edges that will be picked during or after the current iteration. Thus, we need to show that the average number of edges crossing active set S with respect to F' is ≤ 2. The mechanic lies in that every tree or forest, the average degree of vertices is ≤ 2.

Let $H = (V, F')$. Consider the set of connected components with respect to F at the beginning of the current iteration. In H, shrink the set of vertices of each of these components to a single node to obtain H'. All edges picked in F before the current iteration have been shrunk. The degree of a vertex in H' is equal to the degree of the corresponding set in H. Call a vertex of H' an active node if it corresponds to an active component in H. Each active node of H' has nonzero degree (since there must be an edge incident to it to satisfy its requirement), and H' is a forest. Now remove all isolated vertices from H'. The remaining graph is a forest with average degree ≤ 2. By the previous lemma, the degree of each inactive node is ≥ 2, i.e. the forest has no inactive leaves. Hence, the average degree of active nodes is ≤ 2. ■

For each node of degree > 2, there must be correspondingly many active nodes of degree 1.

5.3 Prize-Collecting Steiner Tree

Problem: Prize-Collecting Steiner Tree

Given an undirected graph $G = (V, E)$, edge costs $c_e \geq 0$ for all $e \in E$, a selected root $r \in V$, and penalties $\pi_i \geq 0$ for all $i \in V$, find a tree T that contains the root r so as to minimize the cost of the edges in T plus the penalties of the vertices not in T.

$$\min \sum_{e \in T} c_e + \sum_{i \in V - V(T)} \pi_i$$

We have $x_e \in \{0,1\}$ if e is in T and $z_X \in \{0,1\}$ for all $X \subseteq V-r$ if X is the set of vertices not spanned by T. We need a constraint to enforce that for every subset $S \subseteq V$, either S is a subset of the set of vertices not spanned by T or some of S is being spanned by T, i.e. $\sum_{e\in\delta(S)} x_e + \sum_{X:X\supseteq S} z_X \geq 1$. Since if $x_e = 1$ for $e \in \delta(S)$, then at least some vertex in S is spanned by the tree.

Primal IP

$$\begin{aligned} \min \quad & \sum_{e\in E} c_e x_e + \sum_{X\subseteq V-r} \pi(X) z_X \\ \text{subject to} \quad & \sum_{e\in\delta(S)} x_e + \sum_{X:X\supseteq S} z_X \geq 1, \quad S \subseteq V-r \\ & x_e \in \{0,1\}, \quad e \in E \\ & z_X \in \{0,1\}, \quad X \subseteq V-r \end{aligned}$$

Dual LP

$$\begin{aligned} \max \quad & \sum_{S\subseteq V-r} y_S \\ \text{subject to} \quad & \sum_{S:e\in\delta(S)} y_S \leq c_e, \quad e \in E \\ & \sum_{S:S\subseteq X} y_S \leq \pi(X), \quad X \subseteq V-r \\ & y_S \geq 0, \quad S \subseteq V-r \end{aligned}$$

It can be shown via the max-flow min-cut theorem that in any feasible solution, there is a tree connecting the root r to every vertex i that is not in some set X for which $z_X = 1$.

We design a primal-dual algorithm that grows moats around the vertices in V until they connect to r. However, each moat will come with an associated charge that depletes over time. If the charge is completed, we give up on connecting that moat to r. After we are done, we prune the solution to discard edges not connected to r and edges whose deletion produces a component that had 0 charge at some point in the algorithm.

In the algorithm, we grow a forest F along with a feasible dual y. An active component will be a connected component C of F such that the dual constraint for C has slack. If C contains r, then it is inactive and call this the root component of F.

The component/moat C has charge $\gamma(C)$ that depletes. These charges keep track of the slack of the dual constraint for C. C is active if and only if $\gamma(C) > 0$.

Algorithm: 2-Approximation For Prize-Collecting Steiner Tree

1. Initialize:
   (a) $F \leftarrow \emptyset$
   (b) $\gamma(\{v\}) = \pi_v$ for each $v \in V$
   (c) $\mathcal{C} \leftarrow \{\{v\} : v \in V\}$
   (d) $y \leftarrow 0$
2. **While** there are still active components, do:
   - Simultaneously raise y_C and decrease $\gamma(C)$ for each active component C (at the same rate for each) until one of the following occurs:
     - $\gamma(C)$ becomes 0 for some active component C: do nothing (i.e., C becomes inactive).
     - Some edge e goes tight: Add e to F, set $\gamma(C_1 \cup C_2) \leftarrow \gamma(C_1) + \gamma(C_2)$, where C_1, C_2 are the components of F bridged by e.
       $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_1 \cup C_2\}$.
3. Let C_r denote the root component of F and $F(C_r)$ denote its edges.
4. **While** some $e \in F(C_r)$ is such that the new nonroot component C' of $F - \{e\}$ satisfies $C' \in \mathcal{C}$ and $\gamma(C') = 0$
   - $F \leftarrow F - \{e\}$.
   - Let C_r denote the new nonroot component of this edge set F.
5. Delete every edge e from F that is not in the root component of F.
6. **Return** (F, P), where P is the set of nodes not in the root component of F.

There are a polynomial number of iterations of the first loop since there can be at most n deactivations of an active component between iterations where an edge is added to F. This implies the number of dual variables y_C with positive value constructed is also at most polynomial.

Claim: For any $C, C' \in \mathcal{C}$, we have $C \cap C' = \emptyset$, $C \subseteq C'$, or $C' \subseteq C$ (the positive duals form a laminar family).

Proof of Claim. Suppose C and C' share a vertex v. Then it cannot be that C and C' are different components in an iteration.

Say C is a component in iteration i and C' is a component during iteration j with $i < j$. An easy invariant of this loop is that the components of one iteration are subsets of components of subsequent iterations. So in iteration j, we have that C is a subset of a component in F. Since C contains $v \in C'$ in iteration j, then $C \subseteq C'$. ■

Let (F, P) be the returned solution. We partition the positive dual into two sets

- $\mathcal{S}_1 = \{C \in \mathcal{C} : C \not\subseteq P\}$
- $\mathcal{S}_2 = \{C \in \mathcal{C} : C \subseteq P\}$

In fact, the dual variables for sets in $\mathcal{S}_2$ pay for discarding vertices in P perfectly.

Lemma

$\pi(P) = \sum_{C \in \mathcal{S}_2} y_C$.

Proof. The loop invariant where for each component C of F (active or not) has $\sum_{S \subseteq C} y_S + \gamma(C) = \pi(C)$. Initially this is true because $y = 0$ and $\gamma(\{v\}) \leftarrow \pi(C)$. It holds while raising y_C and decreasing $\gamma(C)$.

Now we show that the loop invariant holds when some edge e bridging C_1 and C_2 is added to F. Not that for any $C \in \mathcal{C}$ with $C \subseteq C_1 \cup C_2$ must be either a subset of C_1 or C_2 by the claim. Thus,

$$\sum_{S \subseteq C_1 \cup C_2} y_S = \sum_{S \subseteq C_1} y_S + \sum_{S \subseteq C_2} y_S = \pi(C_1) - \gamma(C_1) + \pi(C_2) - \gamma(C_2) = \pi(C_1 \cup C_2) - \gamma(C_1 \cup C_2)$$

Now consider $v \in P$. If v was not in the root component of F before the pruning stage, then it lies in a component $C \in \mathcal{C}$ such that $\gamma(C) = 0$. If v was in the root component of F after the first loop but was pruned in the second loop, then it also lies in some $C \in \mathcal{C}$ with $C \subseteq P$ and $\gamma(C) = 0$, so $\sum_{S \subseteq C} y_S = \pi(C)$.

Let $C_1, \ldots, C_k$ be the maximal subsets of $\mathcal{S}_2$ (i.e. $C_i \in \mathcal{S}_2$ but there is no $C' \in \mathcal{S}_2$ such that $C_i \subsetneq C'$). We just showed $P = \bigcup_{i=1}^{k} C_i$ and $\pi(C_i) = \sum_{S \subseteq C_i} y_S$. By the claim, any $S \in \mathcal{S}_2$ must be a subset of some C_i, so $\sum_{S \in \mathcal{S}_2} y_S = \pi(P)$. ■

Lemma

$c(F) \leq 2 \sum_{S \in \mathcal{S}_1} y_S$.

Proof. Let F denote the final set of returned edges. Note that any $C \in \mathcal{C}$ with $\delta(C) \cap F$ must satisfy $C \in \mathcal{S}_1$. This is because no edge of F has an endpoint in P.

Consider some iteration of the first while loop and let F_i denote the set of edges in this iteration. Consider the graph H that consists of a node shrunk for each component of F_i and edges $e \in F$ that bridge the two components. This graph H consists of isolated nodes plus a single tree that includes the root component of F_i. Furthermore, each leaf of H must either be in the root component or an active component of F_i in this iteration, otherwise we would have pruned its parent edge in the pruning stage.

Therefore, all leaves of H except, perhaps, the root component are active components of F_i, meaning the active components/nodes of H have average degree at most 2. Using the same averaging argument and relaxes complementary slackness conditions for Steiner Forest, it shows $c(F) \leq 2 \sum_{S \in \mathcal{S}_1} y_S$. ■

Theorem

The algorithm is finds a solution of cost $c(F) + \pi(P) \leq 2\,\mathrm{OPT}$ for the Prize-Collecting Steiner Tree problem.

Proof. Combining the previous two lemmas

$$c(F) + 2\pi(P) \leq 2 \sum_{S \in \mathcal{S}_1} y_S + 2 \sum_{C \in \mathcal{S}_2} y_C = 2 \sum_{S \subseteq V - r} y_S \leq 2\,\mathrm{OPT}$$

thus, we proved a stronger bound where the cost of F plus 2 times penalty cost is at most $2\,\mathrm{OPT}$.
■

Chapter 6

Iterative Rounding

Lemma (Rank Lemma)

Let $P = \{x : Ax \geq b, x \geq 0\}$ and x be an extreme point solution of P such that $x_i > 0$ for each i. Then any maximal number of linearly independent tight constraints of the form $A_i x = b_i$ for row i of A (i.e. rank of A) equals the number of variables.

6.1 Bipartite Matching

The LP relaxation of bipartite matching is the following

$$\begin{aligned} \max \quad & \sum_{e \in E} w_e x_e & (LP_{bm}(G)) \\ \text{subject to} \quad & \sum_{e \in \delta(v)} \leq 1, \quad \forall v \in V_1 \cup V_2 \\ & x_e \geq 0, \quad \forall e \in E \end{aligned}$$

We can solve this LP in polynomial time since there are a polynomial number of constraints.

Theorem

Given any weight function w, there exists an integral matching M such that $w(M) \geq w^T x$ where x is an optimal solution to $LP_{bm}(G)$.

Theorem

$LP_{bm}(G)$ is integral.

Let $\chi(F)$ for $F \subseteq E$ be the characteristic vector of F. The rank lemma implies the following.

Lemma

Given any extreme point solution x to $LP_{bm}(G)$ such that $x_e > 0$ for each $e \in E$, there exists $W \subseteq V_1 \cup V_2$ such that

- $x(\delta(v)) = 1$ for each $v \in W$.
- The vectors in $\{\chi(\delta(v)) : v \in W\}$ are linearly independent.
- $|W| = |E|$.

Algorithm: Iterative Bipartite Matching Algorithm

1. $F \leftarrow \emptyset$.
2. **While** $E(G) \neq \emptyset$,
   - Find an optimal extreme point solution x to $LP_{bm}(G)$ and remove every e with $x_e = 0$ from G.
   - If there is an edge $e = (u, v)$ with $x_e = 1$, then $F \leftarrow F \cup \{e\}$ and $G \leftarrow G - \{u, v\}$.
3. Return F.

We prove correctness in two steps: 1) Returns a matching of optimal weight if the algorithm always finds an e with $x_e = 0$ or an e with $x_e = 1$. 2) Algorithm will always find an edge.

Claim: If the algorithm, in every iteration, finds an edge e with $x_e = 0$ or an edge e with $x_e = 1$, then it returns a matching F of weight at least the optimal solution to $LP_{bm}(G)$.

Proof of Claim. The proof is by induction on number of iterations. Base case is trivial for one iteration.

If we find an edge e with $x_e = 0$ of the algorithm, then the residual problem is to find a matching in the graph $G' = G - e$. The residual solution x_{res} where x is restricted to $G - e$, is a feasible solution to the LP relaxation of the residual problem. By induction, the algorithm returns a matching $F' \subseteq E(G')$ with weight at least the optimal solution to $LP_{bm}(G')$. Since $w(F') \geq w^T x_{res} = w^T x$, the induction hypothesis holds.

If we find an edge $e = (u, v)$ with $x_e = 1$, then the residual problem is to find a matching which contains e. This is the matching problem on $G' = G - \{u, v\}$. Moreover, x_{res} is a feasible solution to the LP relaxation for the residual problem. Inductively, the algorithm will return a matching F' of weight at least the weight of the optimal solution of $LP_{bm}(G)$, and hence $w(F') \geq w^T x_{res}$. The algorithm returns the matching $F = F' \cup \{e\}$ and we have

$$w(F) = w(F') + w_e \geq w^T x_{res} + w_e = w^T x$$

since $x_e = 1$. Therefore, the weight of the matching returned is at least the weight of the LP solution x. ■

Lemma

Given any extreme point solution x of $LP_{bm}(G)$, there must exist and edge e with $x_e = 0$ or $x_e = 1$.

Proof. Suppose for the sake of contradiction $0 < x_e < 1$ for all $e \in E$. Since x is an extreme point solution, by the lemma (from rank lemma), there exists $W \subseteq V_1 \cup V_2$ such that the constraints to W are linearly independent and right, and $|E| = |W|$.

We claim $\deg(v) = 2$ for each $v \in W$ and $\deg(v) = 0$ for each $v \notin W$. Firstly, $\deg(v) \geq 2$ for each $v \in W$ since $x(\delta(v)) = 1$ and $0 < x_e < 1$ for each $e \in E$. This implies

$$2\,|W| = 2\,|E| = \sum_{v \in V} \deg(v) \geq \sum_{v \in W} \deg(v) \geq \sum_{v \in W} 2 = 2\,|W|$$

This implies the inequalities are equalities and thus, $\deg(v) = 0$ for each $v \notin W$ by the first inequality and $\deg(v) = 2$ for each $v \in W$ by second inequality.

Hence, E is a cycle cover on the vertices in W. Let C be any such cycle with all vertices in W. C is an even cycle since G is bipartite, we also have

$$\sum_{v \in C \cap V_1} \chi(\delta(v)) = \sum_{v \in C \cap V_2} \chi(\delta(v))$$

which contradicts the independence of constraints in condition 2 of the lemma. Therefore, any extreme point solution x to $LP_{bm}(G)$ must have an edge e with $x_e = 0$ or $x_e = 1$. ■

From this lemma, we know the algorithm returns a matching of total weight at least the weight of the LP. This proves the theorem.

6.2 Generalized Assignment Problem

Problem: Generalized Assignment/Scheduling on Unrelated Parallel Machines

Given a set of jobs J and machines M, for each job $j \in J$ and a machine $i \in M$, there is a processing time p_{ij} and cost c_{ij}. Machine i is available for T_i time units and the objective is to assign each job to some machine such that the total cost is minimized and no machine is scheduled for no more than its available time.

Theorem (Shmoys and Tardos)

There exists a polynomial-time algorithm for the generalized assignment problem which returns a solution of cost at most C that uses each machines i for $\leq 2T_i$ time units, where C is the cost of an optimal assignment that uses each machine i for at most T_i time units.

We use the iterative relaxation method to obtain an approximation algorithm for this problem.

We can model this problem as a bipartite matching problem: Let $G = J \cup M$ be a complete bipartite graph with edge costs c_{ij}. The generalized assignment problem can be reduced to finding a subgraph F such that $\deg_F(j) = 1$ for each $j \in J$ and the edge represents which job it is assigned to. The time constraint at machines can be modeled by restricting that $\sum_{e \in \delta(i) \cap F} p_{ij} \leq T_i$ for each machine i.

We can strengthen this model by disallowing certain assignments using the following: If $p_{ij} > T_i$, then no optimal solution assigns job j to i and can remove such edges from G.

We model this using a natural LP relaxation. Observe we do not need to place time constraints for all machines, but a subset $M' \subseteq M$ which is initialized to M.

$$\begin{aligned} \min \quad & \sum_{e=(i,j)\in E} c_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{e\in\delta(j)} x_e = 1, \quad \forall j \in J \\ & \sum_{e\in\delta(i)} p_e x_e \leq T_i, \quad \forall i \in M' \\ & x_e \geq 0, \quad \forall e \in E \end{aligned} \tag{LP_{ga}}$$

Lemma (Characterization of Extreme Points of LP_{ga})

Let x be an extreme point solution to LP_{ga} with $0 < x_e < 1$ for each edge e. Then there exist $J' \subseteq J$ and $M'' \subseteq M'$ such that

- $\sum_{e\in\delta(j)} x_e = 1$ for each $j \in J'$ and $\sum_{e\in\delta(i)} p_e x_e = T_i$ for each $i \in M''$.
- The constraints corresponding to J' and M'' are linearly independent.
- $|J'| + |M''| = |E(G)|$.

In addition to the usual step of picking an integral element in the solution, we identify carefully chosen constraints to relax or remove. The choice is dictated by ensuring the removal will allow us to argue that the final integral solution does not have too much violation; we need to ensure that in the absence of an integral element, such a constraint can always be found to be removed.

Algorithm: Iterative Generalized Assignment Algorithm

1. $F \leftarrow \emptyset, M' \leftarrow M$.
2. **While** $J \neq \emptyset$,
   - Find an optimal extreme point solution x to LP_{ga} and remove every variable $x_{ij} = 0$.
   - If there is a variable with $x_{ij} = 1$, then $F \leftarrow F \cup \{ij\}$, $J \leftarrow J - \{j\}$, $T_i \leftarrow T_i - p_{ij}$.
   - (**Relaxation**) If there is a machine i with $\deg(i) = 1$, or a machine i with $\deg(i) = 2$ and $\sum_{j\in J} x_{ij} \geq 1$, then $M' \leftarrow M' - \{i\}$.
3. Return F.

Lemma

Consider any extreme point solution x to LP_{ga}. One of the following must hold.

- There exists an edge $e \in E$ with $x_e \in \{0, 1\}$.
- There exists a machine $i \in M'$ with $\deg(i) = 1$, or $\deg(i) = 2$ and $\sum_{j\in J} x_{ij} \geq 1$.

Proof. Suppose for the sake of contradiction that both conditions do not hold. By the first two steps of step 2, we have $0 < x_e < 1$ for each edge e. Each job j has degree at least two since

$\sum_{e\in\delta(j)} x_e = 1$ and there is no edge with $x_e = 1$. Moreover, each machine in M' has degree at least two, because the constraints for machines with degree one have been removed in the third step of step 2. The extreme point characterization lemma has that $|E| = |J'| + |M''|$. This implies

$$|J'| + |M''| = |E| \geq \frac{\sum_{j\in J} \deg(j) + \sum_{i\in M'} \deg(i)}{2} \geq \frac{\sum_{j\in J} 2 + \sum_{i\in M'} 2}{2} = |J| + |M'| \geq |J'| + |M''|$$

and all inequalities must hold as equalities. The first inequality implies each machine $i \in M - M'$ has degree 0 and the second inequality implies that each job $j \in J'$ and each machine $i \in M''$ have degree exactly two. The last inequality implies $J = J'$ and $M' = M''$. Therefore, G is a union of cycles, with vertices in $J' \cup M''$ (tight constraints).

Consider any cycle C. The total number of jobs in C is exactly equal to the total number of machines in C. Therefore, since each job $j \in J'$ has $\sum_{i\in M''} x_{ij} = 1$, then summing over all jobs, $\sum_{j\in J'} \sum_{i\in M''} x_{ij} = |J'|$ and swapping the summations gives us that the average amount of jobs assigned to one machine must be at least the average, i.e. there must be a machine i with $\sum_{j\in J'} x_{ij} \geq 1$. Hence, this machine i has degree two and $\sum_{j\in J'} x_{ij} \geq 1$, contradicting that the third step in step 2 cannot be applied ■

Proof of Theorem. First, we prove the algorithm returns an assignment of optimal cost. We claim that at any iteration of the algorithm, the cost of the assignment given by F plus the cost of the current LP solution to LP_{ga} is at most the cost of the initial LP solution. This can be shown by induction on number of iterations. Observe that the claim holds trivially before first iteration. In any iteration, if we assign job j to machine i in second step of step 2, then the cost of F increases by c_{ij} and the current LP solution decreases by $c_{ij}x_{ij} = c_{ij}$ since $x_{ij} = 1$. Hence, the claim holds. If we remove a constraint in the third step of 2, then the cost of F remains the same, while the cost of the LP can only decrease. Hence, the claim holds. Thus, finally when F is a feasible assignment, by induction, the cost of F is $\leq$ cost of initial LP solution.

Finally, we show that machine i is used $\leq 2T_i$ units for each i. Fix any i. We argue the claim: if $i \in M'$, then at any iteration, we must have $T_i' + T_i(F) \leq T_i$, where T_i' is the residual time left on i at this iteration and $T_i(F)$ is the time used by jobs assigned to machine i in F. The proof of this claim follows by an inductive argument the same as above.

Now consider when i is removed from M'. There are two possibilities.

- If there is only one job j in i, then the total processing time at i is $T_i(F)+p_{ij} \leq T_i+p_{ij} \leq 2T_i$, where inequality holds because of the pruning step (where we delete edges assigning a job to i if its processing time exceeded T_i).
- If there are two jobs j_1 and j_2 in i, then let x denote the LP solution when the constraint for i is removed. The total processing time at i is at most

$$\begin{aligned} T_i(F) + p_{ij_1} + p_{ij_2} &\leq T_i - x_{ij_1}p_{ij_1} - x_{ij_2}p_{ij_2} + p_{ij_1} + p_{ij_2} \\ &\leq T_i + (1 - x_{ij_1})p_{ij_1} + (1 - x_{ij_2})p_{ij_2} \\ &\leq T_i + (2 - x_{ij_1} - x_{ij_2})T_i \\ &\leq 2T_i \end{aligned}$$

  because $p_{ij_1}, p_{ij_2} \leq T_i$ again by the pruning step and $x_{ij_1} + x_{ij_2} \geq 1$ by the third step of 2.

This completes the proof. ■

6.3 Minimum Spanning Trees

Problem: Minimum Spanning Tree

Given an undirected graph $G = (V, E)$ and edge costs c_e for all $e \in E$, find a spanning tree of minimum edge cost.

Definition: Spanning Tree

A minimal 1-edge-connected subgraph.

The undirected cut LP of MST requires every pair of vertices has a path connecting them.

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e \\ \text{subject to} \quad & x(\delta(S)) \geq 1, \quad \forall S \subset V \\ & x_e \geq 0, \quad \forall e \in E \end{aligned}$$

There are exponentially many constraints, but we can still solve it in polynomial time if we have a polynomial-time separation oracle. For this problem, we just need to find a cut of total capacity < 1. We can do this by doing a max-flow between every pair of vertices.

Another formulation is the subtour elimination LP. Let $E(S)$ be the set of edges with both endpoints in S. For a spanning tree, there are at most $|S| - 1$ edges in $E(S)$. Insisting this for every set eliminates all potential subtours that can be formed in the LP solution.

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e \\ \text{subject to} \quad & x(E(S)) \leq |S| - 1, \quad \forall \emptyset \neq S \subset V \\ & x(E(V)) = |V| - 1 \\ & x_e \geq 0, \quad \forall e \in E \end{aligned} \tag{LP_{mst}}$$

Theorem

Every extreme point solution to the subtour LP is integral and corresponds to the characteristic vector of a spanning tree.

Theorem

There is a polynomial-time separation oracle for the subtour LP.

Proof. Given a fractional solution x, the separation oracle needs to find a set $S \subseteq V$ such that $x(E(S)) > |S| - 1$ if such a set exists. It is easy to check the equality constraint $x(E(V)) = |V| - 1$. Checking for the inequality for each subset is equivalent to checking $\min_{S \subset V}\{|S| - 1 - x(E(S))\} < 0$. Using the equality condition, we obtain that it is enough to check $\min_S\{|S| - 1 + x(E(V)) - x(E(S))\} < |V| - 1$. We show that solving $2|V| - 2$ min-cut problems suffice.

Fix a root $r \in V$. For each $k \in V - r$, we construct two min cut instances, one which checks the inequality for all subsets S containing r but not k, and the other checks the inequality for all subsets S containing k but not r.

Construct a directed graph $\hat{G} = (V, \hat{E})$ where $\hat{E}$ has all arcs (i, j) and (j, i) for each $(i, j) \in E$. Let weight of (i, j) and (j, i) be $x_{ij}/2$. We also place arcs from each $v \in V - \{r, k\}$ to k of weight 1 and arcs from r to each vertex $v \in V - r$ of weight $\sum_{e \in \delta(v)}(x_e/2) = x(\delta(v))/2$.

Consider any cut $(S, V - S)$ which separates r from k. Edges of weight one contribute exactly $|S| - 1$. The edges between i and j of weight $x_{ij}/2$ contribute exactly $x(\delta(S))/2$. The edges from r to the rest of the vertices contribute $\sum_{v \notin S} x(\delta(v))/2$. The total weight of the cut is exactly

$$|S| - 1 + \frac{x(\delta(S))}{2} + \sum_{v \notin S} \frac{x(\delta(v))}{2} = |S| - 1 + x(E(V)) - x(E(S))$$

Hence, checking whether the min cut separating r from k is strictly smaller than $|V| - 1$ checks exactly whether there is a violating set S containing r but not containing k. ■

Lemma

The feasible integer solutions are precisely the $\{0, 1\}$ solutions corresponding to spanning trees of G.

Proof. Let x be a feasible integer solution. Note that $x_{uv} \leq 1$ for each $(u, v) \in E$ because $x(E(S)) \leq 1$ is satisfied for $S = \{u, v\}$. Let $T = \{e : x_e = 1\}$. $|T| = x(E(V)) = |V| - 1$. Furthermore, T cannot contain a cycle since if T contained a cycle with vertices C, then $x(E(C)) \geq n$, which contradicts feasibility of x. Any graph on $|V|$ vertices that has $|V| - 1$ edges and does not contain a cycle is a spanning tree.

Conversely, any spanning tree T contains exactly $|V| - 1$ edges and for each $S \subseteq V$, at most $|S| - 1$ edge of T have both endpoints in S, otherwise there is a cycle in S. So the integer solution to T is a solution in LP_{mst}. ■

6.3.1 Uncrossing Technique

We will use the property of laminar sets to show LP_{mst} is integral.

Definition: Cross

Two sets $A, B \subseteq V$ cross if $A \cap B \neq \emptyset$, but neither is a subset of the other.

Definition: Laminar

A family $\mathcal{L}$ of subsets of V is called laminar if no two of its subsets cross, i.e. for $A, B \in \mathcal{L}$, we have either $A \cap B = \emptyset$, $A \subseteq B$, or $B \subseteq A$.

Lemma

Let $\mathcal{L}$ be a laminar family of V such that $|A| \geq 2$ for any $A \in \mathcal{L}$. Then $|\mathcal{L}| \leq |V| - 1$.

Theorem

LP_{mst} is integral.

Proof. Let x be an extreme point. x is an extreme point if and only if the corresponding solution after deleting e with $x_e = 0$ is an extreme point. So we assume $x_e > 0$ for each $e \in E$.

We want to show that $|E| \leq |V| - 1$. If so, we are done since we have $x(E(V)) = |V| - 1$ and $x_e \leq 1$ for each $e \in E$. So, $x_e = 1$ for each $e \in E$.

By the properties of extreme points, $|E|$ is equal to the rank of the collection of vectors $\mathcal{M} = \{\chi(E(S)) : x(E(S)) = |S| - 1\}$, i.e. the tight constraints. We show that there is a laminar family $\mathcal{L}$ consisting only of S with $|S| \geq 2$ and $x(E(S)) = |S| - 1$ such that the vectors $\chi(E(S)), S \in \mathcal{L}$ form a basis of tight constraints. If so, by the previous lemma,

$$|E| = \text{rank}(\mathcal{M}) = \text{rank}(\{\chi(E(S)) : S \in \mathcal{L}\}) = |\mathcal{L}| \leq |V| - 1$$

Let $\mathcal{L}$ be the largest laminar collection of V such that $\chi(E(S))$ are linearly independent. If $|\mathcal{L}| < |E|$, then there is some $R \subseteq V, |R| \geq 2$ such that $x(E(R)) = |R| - 1$, but $\chi(E(R)) \notin \text{span}\{\chi(E(S)) : S \in \mathcal{L}\}$.

Because R cannot be added to $\mathcal{L}$, we know that R crosses $S \in \mathcal{L}$. Choose such an R that crosses the fewest sets in $\mathcal{L}$ and let $S \in \mathcal{L}$ be any set that crosses R.

Proposition

For $X, Y \subseteq V$,

$$\chi(E(X)) + \chi(E(Y)) = \chi(E(X \cup Y)) + \chi(E(X \cap Y)) - \chi(E(X - Y, Y - X))$$

Let $F' = E(S - R, R - S)$ be the set of edges with one endpoint in each set difference. Then we have

$$\begin{aligned}
|R| - 1 + |S| - 1 &= x(E(R)) + x(E(S)) && \text{(corresponding constraints are tight)} \\
&= x(E(R \cup S)) + x(E(R \cap S)) - x(F') && \text{(by proposition)} \\
&\leq |R \cap S| - 1 + |R \cup S| - 1 && (x \text{ is feasible}) \\
&= |R| - 1 + |S| - 1
\end{aligned}$$

Therefore, all inequalities hold with equality, i.e. $x(E(S \cup R)) = |S \cup R| - 1$, $x(E(S \cap R)) = |S \cap R| - 1$, and $x(F') = 0$.

Because $x_e > 0$, then $F' = \emptyset$ which means $x(E(R)) + x(E(S)) = x(E(R \cup S)) + x(E(R \cap S))$. Both $R \cup S$ and $R \cap S$ can only cross sets in $\mathcal{L}$ that R crossed. Since both do not cross S, then both cross fewer sets in $\mathcal{L}$ than R.

Finally, it cannot be that both $\chi(E(R \cup S)), \chi(E(R \cap S)) \in \text{span}\{\chi(E(S')) : S' \in \mathcal{L}\}$, otherwise $\chi(E(R)) = \chi(E(R \cup S)) + \chi(E(R \cap S)) - \chi(E(S)) \in \text{span}\{\chi(E(S')) : S' \in \mathcal{L}\}$. Therefore, at least one of $R' \in \{R \cap S, R \cup S\}$ is such that $\chi(E(R')) \notin \text{span}\{\chi(E(S')) : S' \in \mathcal{L}\}$, R' crosses fewer sets of $\mathcal{L}$ than R (from previous paragraph), and the constraint for R' is tight, a contradiction. ■

Algorithm: Iterative Leaf-Finding MST

1. $F \leftarrow \emptyset$.
2. **While** $|V(G)| \geq 2$,
   - Find an optimal extreme point solution x to the subtour LP and remove every edge e with $x_e = 0$ from G.
   - Find a vertex v with at most one edge $e = uv$ incident to it, then $F \leftarrow F \cup \{e\}$ and $G \leftarrow G - v$.
3. Return F.

6.4 Minimum Bounded Degree Spanning Trees

Problem: Minimum Bounded-Degree Spanning Tree

Given a graph $G = (V, E)$, edge costs c_e for all $e \in E$, and a degree upper bound B_v for each $v \in V$, find a minimum cost spanning tree which satisfies degree bounds.

Theorem

There exists a polynomial-time algorithm which given a feasible instance of MBDST returns a spanning tree T such that $\deg_T(v) \leq B_v + 1$ and cost of T is at most the cost of any tree which satisfies the degree bounds.

The algorithm involves no rounding or picking an edge with value 1. It removes degree constraints one by one and reduces the problem to a MST problem.

In the following LP relaxation, we assume degree bounds are given for vertices only in a subset $W \subseteq V$. Let $\mathcal{B}$ denote the vector of all degree bounds B_v for each $v \in W$.

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e && (LP_{mbdst}(G, \mathcal{B}, W)) \\ \text{subject to} \quad & x(E(V)) = |V| - 1 \\ & x(E(S)) \leq |S| - 1, \quad \forall \emptyset \neq S \subset V \\ & x(\delta(v)) \leq B_v, \quad \forall v \in W \\ & x_e \geq 0, \quad \forall e \in E \end{aligned}$$

Lemma

Let x be an extreme point to $LP_{mbdst}(G, \mathcal{B}, W)$ with $x_e > 0$ for each $e \in E$. Then there exists a set $T \subseteq W$ and a laminar family $\mathcal{L}$ such that

- $x(\delta(v)) = B_v$ for each $v \in T$ and $x(E(S)) = |S| - 1$ for each $S \in \mathcal{L}$.
- The vectors in $\{\chi(E(S)) : S \in \mathcal{L}\} \cup \{\chi(\delta(v)) : v \in T\}$ are linearly independent.
- $|\mathcal{L}| + |T| = |E|$.

Algorithm: Iterative Relaxation MBDST

1. **While** $W \neq \emptyset$,
   - Find an optimal extreme point solution x to $LP_{mbdst}(G, \mathcal{B}, W)$ and remove every edge e with $x_e = 0$ from G.
   - $E \leftarrow \{e \in E : x_e > 0\}$
   - (**Relaxation**) If $|\delta(v) \cap E| \leq B_v + 1$, then $W \leftarrow W - \{v\}$.
2. Return E.

We prove that in each iteration, the algorithm can find some vertex for which the degree constraint can be removed. Observe that once all degree constraints are removed, we obtain the LP for MST which is integral. Hence, the algorithm returns a tree. Moreover, in each iteration, we only relax the LP, so the cost of the final solution is at most the cost of the initial LP solution. Thus, the tree has optimal cost. An inductive argument also shows that the degree bound is violated by at most one. The degree bound is violated only when we remove the degree constraint and then $|\delta(v) \cap E| \leq B_v + 1$. Thus, in the worst case, if we include all edges incident to v in T, the degree bound of v is violated by at most one.

It remains to show that the iterative relaxation finds a degree constraint to remove at each step. From the lemma, we have that there exists a laminar family $\mathcal{L} \subseteq \mathcal{F}$ and $T \subseteq W$ such that $|\mathcal{L}| + |T| = |E|$. Observe that if $T = \emptyset$, then only the spanning tree inequalities define the solution x. Hence, x must be integral. In the other case, we show that there must be a vertex in W whose degree constraint can be removed.

Lemma

Let x be an extreme point to $LP_{mbdst}(G, \mathcal{B}, W)$ with $x_e > 0$. Let $\mathcal{L}$ and $T \subseteq W$ correspond to the tight set constraints and the tight degree constraints defining x by the lemma. If $T \neq \emptyset$, then there exists some $v \in W$ with $\deg_E(v) \leq B_v + 1$.

Proof. We use the local fractional token argument. Suppose for the sake of contradiction, we have $T \neq \emptyset$ and $|\delta(v) \cap E| \geq B_v + 2$ for each $v \in W$. Assign 1 charge for each $e \in E$. We then redistribute the charge such that each vertex in T and each set in $\mathcal{L}$ gets one charge and we still have extra charges left. This will contradict $|E| = |T| + |\mathcal{L}|$.

The charge redistribution is as follows: each edge $e \in E$ gives x_e units of charge to the smallest $S \in \mathcal{L}$ with $u, v \in S$. Send $\frac{1-x_e}{2}$ units of charge to each u and v that lies in T. Note that since e sends out at most 1 unit of charge, the total charge sent out is at most $|E|$ (we show it is strictly less than $|E|$).

We need to show that each $v \in T$ and each $S \in \mathcal{L}$ collects at least 1 unit of charge. Consider some $v \in T$. The total charge that v collects is precisely

$$\sum_{e \in \delta(v)} \frac{1 - x_e}{2} = \frac{|\delta(v) \cap E| - B_v}{2} \geq 1$$

where the first equality holds since $\sum_{e \in \delta(v)} x_e = B_v$ and the inequality holds since $|\delta(v) \cap E| \geq B_v + 2$ by assumption.

Now consider some $S \in \mathcal{L}$. Let $R_1, \ldots, R_k$ denote the maximal subsets of S in $\mathcal{L}$. That is, each

$R_i \in \mathcal{L}$ is a proper subset of S and no other $R' \in \mathcal{L}$ satisfies $R_i \subsetneq R' \subsetneq S$. The total charge that S collects is precisely the sum of all edges that are contained in S subtracting all edges that are contained in the maximal R_i's (by definition of charging),

$$x(E(S)) - \sum_{i=1}^{k} x(E(R_i)) = (|S| - 1) - \sum_{i=1}^{k} (|R_i| - 1)$$

Also, $|R_1| + \cdots + |R_k| \leq |S|$, so the RHS is nonnegative. Furthermore, $\chi(E(S)) \neq \sum_i \chi(E(R_i))$ (by linear independence of $\mathcal{L}$), so there is some edge $e \in E$ in $E(S)$ but not in any $E(R_i)$. The expression for the total charge assigned to S must be integral and positive, implying it collects at least 1 unit of charge.

We now need to argue there is some extra charge left. There are two simple cases to consider first.

- If $V \notin \mathcal{L}$, then there exists an edge e which is not contained any set of $\mathcal{L}$ and the x_e charge for that edge gives a contradiction.
- If there is a $v \in W - T$, then each edge e incident at v must have $x_e = 1$, otherwise the $(1 - x_e)/2 > 0$ is extra.

Assume neither of these occur. Note that if $x_e = 1$ for some $e \in E$, then $\chi(e) \in \text{span}(\{\chi(E(S)) : S \in \mathcal{L}\})$, since e is a tight set of size two. Now,

$$2\chi(E(V)) = \sum_{v \in V} \chi(\delta(v)) = \sum_{v \in T} \chi(\delta(v)) + \sum_{v \in V - T} \chi(\delta(v)) = \sum_{v \in T} \chi(\delta(v)) + \sum_{v \in V - T} \sum_{e \in \delta(v)} \chi(e)$$

Since $T \neq \emptyset$, then the first summation is nonzero. Now since $\chi(e)$ is in the span of $\{\chi(E(S)) : S \in \mathcal{L}\}$ and that $V \in \mathcal{L}$, then we just wrote the vectors $\{\chi(\delta(v)) : v \in T\}$ by a linear combination of the vectors in $\{\chi(E(S)) : S \in \mathcal{L}\}$, contradicting the fact that the vectors in $\{\chi(E(S)) : S \in \mathcal{L}\} \cup \{\chi(\delta(v)) : v \in T\}$ are linearly independent. ■

6.5 Survivable Network Design Problem

Definition: Steiner Network

A subgraph of G in which there are at least r_{uv} edge-disjoint paths between u and v for every pair $u, v \in V$.

Problem: Survivable Network Design

Given an undirected graph $G = (V, E)$ and a connectivity requirement r_{uv} for each pair $u, v \in V$, find a minimum-cost Steiner network.

Definition: Skew Supermodular

A function $f : 2^V \to \mathbb{Z}$ is skew supermodular if at least one of the following conditions hold for any two $S, T \subseteq V$.

$$f(S) + f(T) \leq f(S \cup T) + f(S \cap T)$$
$$f(S) + f(T) \leq f(S \setminus T) + f(T \setminus S)$$

It can be shown that $f(S) = \max_{u \in S, v \notin S}\{r_{uv}\}$ for each $S \subseteq V$ is a skew supermodular function. The LP relaxation for the survivable network design problem is then

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_E \\ \text{subject to} \quad & x(\delta(S)) \geq f(S), \quad \forall S \subseteq V \\ & 0 \leq x_e \leq 1, \quad \forall e \in E \end{aligned} \tag{LP_{sndp}}$$

It is not known whether there is a polynomial-time separation oracle for any skew supermodular f, but this problem can be solved using a max flow separation oracle.

For a subset $S \subseteq V$, $x(\delta(S)) \geq f(S)$ defines a characteristic vector $\chi(\delta(S))$ in $\mathbb{R}^{|E|}$. By strong submodularity of the cut function in undirected graphs, we have

$$x(\delta(X)) + x(\delta(Y)) \geq x(\delta(X \cup Y)) + x(\delta(X \cap Y))$$

and

$$x(\delta(X)) + x(\delta(Y)) \geq x(\delta(X \setminus Y)) + x(\delta(Y \setminus X))$$

When f is skew supermodular, it follows from uncrossing arguments that an extreme point to LP_{sndp} is characterized by a laminar family of tight constraints.

Lemma

Let the requirement function f of LP_{sndp} be skew supermodular, and let x be an extreme point solution with $0 < x_e < 1$ for every $e \in E$. Then there exists a laminar family $\mathcal{L}$ such that

- $x(\delta(S)) = f(S)$ for each $S \in \mathcal{L}$.
- The vectors in $\{\chi(\delta(S)) : S \in \mathcal{L}\}$ are linearly independent.
- $|E| = |\mathcal{L}|$.

Algorithm: Jain's Iterative Minimum Steiner Network

1. $F \leftarrow \emptyset, f' \leftarrow f$.
2. **While** $f' \not\equiv 0$,
   - Find an optimal extreme point solution x to LP_{sndp} with cut requirement f' and remove every e with $x_e = 0$.
   - If there exists an edge e with $x_e \geq \frac{1}{2}$, $F \leftarrow F \cup \{e\}$ and $G \leftarrow G - \{e\}$.
   - For every $S \subseteq V$, $f'(S) \leftarrow \max\{f(S) - |\delta(S) \cap F|, 0\}$.
3. Return $H = (V, F)$.

Theorem

Suppose f is a skew supermodular function and x is an extreme point solution to LP_{sndp}. Then there exists an edge $e \in E$ with $x_e \geq \frac{1}{2}$.

Assuming this theorem, the iterative algorithm will terminate and it can be shown by an inductive argument that the returned solution is a 2-approximation. We prove this theorem by a counting argument.

Proof. Suppose for a contradiction that every edge e has $0 < x_e < \frac{1}{2}$. By the lemma, there is a laminar family $\mathcal{L}$ of tight constraints that define x. We assign 2 charges to each edge, one to each endpoint, for a total of $2|E|$ charges. We redistribute the charges so that each member $\mathcal{L}$ receives ≥ 2 charges and there are some charges left. This would imply $|E| > |\mathcal{L}|$, contradicting $|E| = |\mathcal{L}|$.

Let $S \subseteq V$. Define its co-requirement $coreq(S) := \sum_{e \in \delta(S)} \left(\frac{1}{2} - x_e\right)$. We call a set $S \in \mathcal{L}$ special if $coreq(S) = \frac{1}{2}$. Given a laminar family, it can be represented as a forest of trees if its sets are ordered by inclusion. Recall that in this forest, a set $R \in \mathcal{L}$ is a child of $S \in \mathcal{L}$ if S is the smallest set in $\mathcal{L}$ that contains R. We say an endpoint v is owned by S if S is the smallest set in $\mathcal{L}$ that contains v. The following lemma is useful to establish a certain set is special.

Lemma $(*)$

Suppose S has α children and owns β endpoints where $\alpha + \beta = 3$. If all children of S are special, then S is special.

The redistribution of tokens is by an inductive argument based on the following lemma.

Lemma

For any rooted subtree of the forest $\mathcal{L}$ with root S, the charges assigned to vertices in S can be redistributed such that every member in the subtree gets at least 2 charges, and the root S gets at least 3 charges. Furthermore, S gets exactly 3 charges only if S is special; otherwise S gets at least 4 charges.

Proof of Lemma. Proof by induction. For the base case, consider a leaf node S in the laminar family. Since $f(S) \geq 1$ and $x_e < \frac{1}{2}$ for all e, this implies $\deg(S) \geq 3$ and thus S can collect 3 charges. Furthermore, S collects exactly 3 charges only if $coreq(S) = \deg(S)/2 - x(\delta(S))$. In this case, $f(S) = 1$ and $\deg(S) = 3$, and thus S is special.

For inductive step, consider a non-leaf node S. For a child R of S, we say R has one excess charge if R has 3 charges, and R has at least two excess charges if R has at least 4 charges.

By the inductive hypothesis, each child has at least one excess charge, and has exactly one excess charge only if it is special. We divide the cases by the number of children of S.

(i) S has ≥ 4 children: Then S can collect 4 charges by taking one excess charge from each child.

(ii) S has 3 children: If any child of S has 2 excess charges or if S owns any endpoint, then S can collect 4 charges. Otherwise, all 3 children of S are special and S does not own any endpoint, but then S is special by lemma $(*)$, and so 3 charges are enough for the inductive hypothesis.

(iii) S has 2 children: If both children have 2 excess charges, then S can collect 4 charges. So assume that one child of S is special, but then it can be shown that S must own at least one endpoint. If both children of S are special, then S is special by lemma $(*)$, and so 3 tokens are enough for the inductive hypothesis. Otherwise, S can collect 4 charges.

(iv) S has one child R: Since $\chi(\delta(S))$ and $\chi(\delta(R))$ are linearly independent, S must own at least one endpoint. As both $f(S)$ and $f(R)$ are integers and there is no edge of integral value, this

implies that S cannot own exactly one endpoint, and thus S owns at least 2 endpoints. If R is not special, then S can collect 4 charges; otherwise, S is special by lemma $(*)$ and so 3 tokens are enough for the inductive hypothesis.

■

The extra charges in the roots of the laminar family give the contradiction. This proves the theorem and the 2-approximation algorithm. ■

Chapter 7

Tree Metrics

Let d be a metric over points V and d' be a metric over points W.

Definition: Embedding

A mapping $\phi : V \to W$.

Without loss of generality, the embedding does not shrink distances.

Definition: Distortion

An embedding has distortion α if

$$d_{uv} \leq d'_{\phi(u)\phi(v)} \leq \alpha \cdot d_{uv}$$

Definition: Tree Metric

A tree metric (V', T) for vertices V is a tree T defined on $V' \supseteq V$, together with nonnegative lengths on each edge of T.

The distance T_{uv} for $u, v \in V'$ is the length of the unique shortest uv-path in T.

We would like to have a tree metric (V', T) that approximates d on V with $d_{uv} \leq T_{uv} \leq \alpha \cdot d_{uv}$ for all $u, v \in V$ for some distortion α.

Given a low-distortion embedding, we want to produce an algorithm on a tree metric than the general metric since it is often easier. Unfortunately, it can be shown for a cycle on n vertices, no tree metric has distortion less than $(n-1)/8$. However, we can give a randomized algorithm for producing a tree T such that $d_{uv} \leq T_{uv}$ and $E[T_{uv}] \leq O(\log n) d_{uv}$ (expected $O(\log n)$ distortion). This is the probabilistic approximation of metric d by a tree metric.

7.1 Probabilistic Approximation of Metrics

Theorem

Given a distance metric (V, d) such that $d_{uv} \geq 1$ for all $u, v \in V, u \neq v$, there is a randomized, polynomial-time algorithm that produces a tree metric $(V', T), V \subseteq V'$, such that for all $u, v \in V$, $d_{uv} \leq T_{uv}$ and $E[T_{uv}] \leq O(\log n) d_{uv}$.

The tree is constructed via a *hierarchical cut decomposition* of metric d. Let Δ be the smallest power of 2 greater than $2 \max_{u,v} d_{uv}$ (the diameter of V). That is, $\Delta = 2^h$ where $h = \lceil \log_2(2 \max_{u,v} d_{uv}) \rceil$. The hierarchical cut decomposition is a rooted tree with $\log_2 \Delta + 1$ levels and is built as follows:

- The nodes at each level of the tree correspond to a partitioning of V. For a node corresponding to $S \subseteq V$, then
$$S = \bigcup_{1 \leq i \leq k} S_k \text{ and } S_i \cap S_j = \emptyset, i \neq j$$
where $S_1, \ldots, S_k$ are the children nodes of S.
- For level i, the vertices in S will be the vertices will be in a ball of radius $< 2^i$ and $\geq 2^{i-1}$ centered on some vertex.
- The length of the tree edge connecting level $i - 1$ to a parent at level i is 2^i.

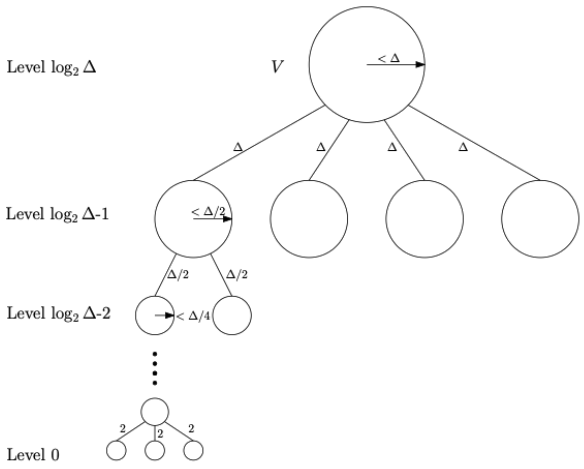


Figure 7.1: Hierarchical Cut Decomposition

Observe by definition of Δ, the set V is contained in a ball of radius Δ centered on any vertex, since the radius of the ball at level $\log_2 \Delta$ is at least $\frac{1}{2}\Delta \geq \max_{u,v} d_{uv}$. Note each leaf at level 0 is in a ball of radius $< 2^0 = 1$ centered on some u, and so u is in the ball by itself since $d_{uv} \geq 1$ for all $v \neq u$.

Each node of the tree will be a vertex in V', so that the tree is a tree on the vertices in V'. We let each leaf correspond to the unique $v \in V$, so that the leaves are exactly V, while the internal vertices are the remaining nodes in V'.

Lemma

Any tree T obtained via the hierarchical cut decomposition of metric d as above has $T_{uv} \geq d_{uv}$ for all pairs $u, v \in V$. Furthermore, if the least common ancestor of $u, v \in V$ is at level i, then $T_{uv} \leq 2^{i+2}$.

Proof. d_{uv} for any pair $u, v \in S$ corresponding to a node at level i is $< 2^{i+1}$, since the radius of the ball containing S must be less than 2^i. Thus, vertices u, v cannot belong to the same node at level $\lfloor \log_2 d_{uv} \rfloor - 1$, since otherwise the distance between them would be less than $2^{\lfloor \log_2 d_{uv} \rfloor} \leq d_{uv}$, a contradiction.

The lowest level at which u and v can belong to the same node is then $\lfloor \log_2 d_{uv} \rfloor$. Therefore, the distance

$$T_{uv} \geq 2 \sum_{j=1}^{\lfloor \log_2 d_{uv} \rfloor} s^j \geq d_{uv}$$

since the length of the tree edge joining level $j-1$ to j is 2^j and the path from u to v starts at u at level 0, goes through a node of level at least $\lfloor \log_2 d_{uv} \rfloor$ and back to v at level 0.

If the least common ancestor of $u, v \in V$ is at level i, then

$$T_{uv} = 2 \sum_{j=1}^{i} 2^j = 2^{i+2} - 4 \leq 2^{i+2}$$

■

The randomized algorithm begins by picking a random permutation π of the vertices and setting a radius r_i for all balls at level i. We pick $r_0 \in [1/2, 1)$ uniformly at random and set $r_i = 2^i r_0$ for $1 \leq i \leq \log_2 \Delta$. Note that any r_i is distributed in $[2^{i-1}, 2^i)$ uniformly.

To produce the tree metric, the children of a node at level $i-1$ corresponding to S at level i is obtained by going through all of V in the order of the permutation starting with $\pi(1)$. For a vertex $\pi(j)$, we consider the ball $B(\pi(j), r_{i-1})$: if $B(\pi(j), r_{i-1}) \cap S = \emptyset$, we go on to vertex $\pi(j+1)$, otherwise we make $B(\pi(j), r_{i-1}) \cap S$ a child node of S, remove vertices of $B(\pi(j), r_{i-1})$ from S, and go onto vertex $\pi(j+1)$ with the remaining nodes of S, if any.

All vertices of S are accounted for since every vertex in S is a member of the centered on itself. Observe also that a child node of S can have as its center a vertex $\pi(j)$ not in S, or a vertex $\pi(j)$ that is in another previously formed part of the partition.

Algorithm: Randomized Tree Metric

1. Pick a random permutation π of V.
2. Set Δ to smallest power of 2 greater than $2 \max_{u,v} d_{uv}$.
3. Pick $r_0 \in [1/2, 1)$ uniformly at random. Set $r_i = 2^i r_0$ for all $1 \leq i \leq \log_2 \Delta$.
4. $\mathcal{C}(\log_2 \Delta) = \{V\}$ ($\mathcal{C}(i)$ will be sets corresponding to nodes at level i that partition V).
5. **For** $i \leftarrow \log_2 \Delta$ to 1,
   - $\mathcal{C}(i-1) \leftarrow \emptyset$.
   - **For** all $C \in \mathcal{C}(i)$,
     - $S \leftarrow C$.
     - **For** $j \leftarrow 1$ to n,
       - If $B(\pi(j), r_{i-1}) \cap S \neq \emptyset$, then add $B(\pi(j), r_{i-1}) \cap S$ to $\mathcal{C}(i-1)$, and remove $B(\pi(j), r_{i-1}) \cap S$ from S.
     - Create tree nodes corresponding to all sets in $\mathcal{C}(i-1)$ that are subsets of C.
     - Join these nodes to node corresponding to C by an edge of length 2^i.

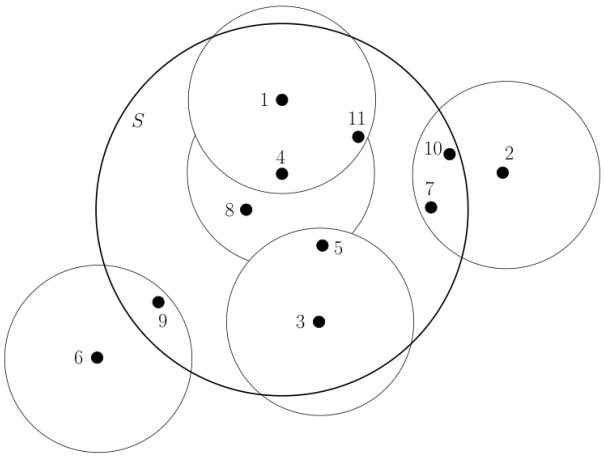


Figure 7.2: Consider $S = \{1, 3, 4, 5, 7, 8, 9, 10, 11\}$ and identity permutation. The figure is a partitioning of S for the child nodes.

Proof of Theorem 7.1. The lemma shows that for a tree T, $T_{uv} \geq d_{uv}$.

To show the other inequality, pick a particular pair $u, v \in V$. The lemma also shows that the length of T_{uv} depends on the level of the least common ancestor of u and v, and if this level is level $i+1$, $T_{uv} \leq 2^{i+3}$. For this least common ancestor to be at level $i+1$, u and v must be in different sets on level i. In order for this to happen, there must be some vertex w such that exactly one of u and v is in the set corresponding to the ball centered on w on level i.

We say w settles the pair u, v on level i if w is the first vertex in the random permutation of vertices such that at least one of u, v is in the ball $B(w, r_i)$. We say w cuts the pair u, v on level i if exactly one of u and v is in $B(w, r_i)$. Let X_{iw} be the event that w cuts (u, v) on level i and let S_{iw} be the

event that w settles (u, v) on level i. Then if $\mathbf{1}$ is the indicator function,

$$T_{uv} \leq \max_{i=0,\ldots,\log\Delta-1} \mathbf{1}(\exists w \in V : X_{iw} \wedge S_{iw}) \cdot 2^{i+3}$$

We can simplify slightly by replacing the maximum and existential quantifier with sums,

$$T_{uv} \leq \sum_{w\in V} \sum_{i=1}^{\log_2\Delta-1} \mathbf{1}(X_{iw} \wedge S_{iw}) \cdot 2^{i+3}$$

Taking the expectation,

$$E[T_{uv}] \leq \sum_{w\in V} \sum_{i=0}^{\log_2\Delta-1} \Pr[X_{iw} \wedge S_{iw}] \cdot 2^{i+3}$$

We give an upper bound b_w on $\Pr[S_{iw}|X_{iw}]$ that depends only on w and show that $\sum_{i=1}^{\log_2\Delta-1} \Pr[X_{iw}] \cdot 2^{i+3} \leq 16d_{uv}$. Assume this, then

$$\begin{aligned}
E[T_{uv}] &\leq \sum_{w\in V} \sum_{i=0}^{\log_2\Delta-1} \Pr[X_{iw} \wedge S_{iw}] \cdot 2^{i+3} \\
&= \sum_{w\in V} \sum_{i=0}^{\log_2\Delta-1} \Pr[S_{iw}|X_{iw}] \Pr[X_{iw}] \cdot 2^{i+3} \\
&\leq \sum_{w\in V} b_w \sum_{i=0}^{\log_2\Delta-1} \Pr[X_{iw}] \cdot 2^{i+3} \\
&\leq 16d_{uv} \sum_{w\in V} b_w
\end{aligned}$$

In particular, we want to show $\sum_{w\in V} b_w = O(\log n)$. First we show that $\sum_{i=1}^{\log_2\Delta-1} \Pr[X_{iw}] \cdot 2^{i+3} \leq 16d_{uv}$. Suppose without loss of generality that $d_{uw} \leq d_{vw}$. Then the probability that w cuts (u, v) on level i is the probability that $u \in B(w, r_i)$ and $v \notin B(w, r_i)$, or that $d_{uw} \leq r_i < d_{vw}$. Since $r_i \in [2^{i-1}, 2^i)$ uniformly at random, this probability is simply $\frac{1}{2^i-2^{i-1}}$ times the length of the intersection of the intervals $[2^{i-1}, 2^i)$ and $[d_{uw}, d_{vw})$, so that

$$\Pr[X_{iw}] = \frac{\left|[2^{i-1}, 2^i) \cap [d_{uw}, d_{vw})\right|}{|[2^{i-1}, 2^i)|} = \frac{\left|[2^{i-1}, 2^i) \cap [d_{uw}, d_{vw})\right|}{2^{i-1}}$$

Then

$$2^{i+3} \Pr[X_{iw}] = \frac{2^{i+3}}{2^{i-1}} \cdot \left|[2^{i-1}, 2^i) \cap [d_{uw}, d_{vw})\right| = 16 \left|[2^{i-1}, 2^i) \cap [d_{uw}, d_{vw})\right|$$

Since the intervals $[2^{i-1}, 2^i)$ for $i = 0$ to $\log_2\Delta - 1$ partition the interval $\left[\frac{1}{2}, \frac{\Delta}{2}\right)$, it follows that

$$\sum_{i=0}^{\log_2\Delta-1} \Pr[X_{iw}] \cdot 2^{i+3} \leq 16\,|[d_{uw}, d_{vw})| = 16(d_{vw} - d_{uw}) \leq 16d_{uv}$$

where the final inequality comes from triangle inequality.

To bound $\Pr[S_{iw}|X_{iw}]$, we order the vertices $w \in V$ in order of their distance to the pair u, v, that is, order $w \in V$ by $\min(d_{uw}, d_{vw})$. Note that if X_{iw} occurs, then one of u and v is in the ball

$B(w, r_i)$. Thus, any vertex z closer to the pair u, v than w will also have at least one of u and v in the ball $B(z, r_i)$. So in order for w to settle the pair u, v given that it cuts u, v on level i, it must come before all closer vertices z in the random permutation. If w is the jth closest vertex u, v, then out of the $j!$ possible permutations of the first j vertices, exactly $(j-1)!$ of them have w first. Thus, this happens with probability $\leq 1/j$. We can then define the bound b_w on this probability as $1/j$. Since for each $1 \leq j \leq n$, there is some vertex w that is the jth closest to the pair u, v, we have that

$$\sum_{w \in V} b_w = \sum_{j=1}^{n} \frac{1}{j} = O(\log n)$$

■

7.2 Buy-at-Bulk Network Design

Problem: Buy-at-Bulk Network Design

Given an undirected graph $G = (V, E)$ with lengths $\ell_e \geq 0$ for each $e \in E$, k source-sink pairs $s_i, t_i \in V$, and each pair has an associated demand d_i. We can purchase a capacity u on any edge at cost $f(u)$ per unit distance, where $f(0) = 0$, f is nondecreasing, and that f obeys economies of scale (cost per unit of capacity does not increase as desired capacity increases, i.e. f is subadditive: $f(u_1 + u_2) \leq f(u_1) + f(u_2)$.
The goal is to find a path P_i from s_i to t_i for each i and a minimum-cost set of capacities c_e for all $e \in E$ such that a multicommodity flow can be routed in the graph of non-zero capacities.
That is, we can send d_i units of commodity i from s_i to t_i along P_i using the capacities c_e. The cost of the solution is

$$\sum_{e \in E} f(c_e)\ell_e$$

Observe the problem is easy to solve in a tree metric T. Because there is only one path P_i in T between s_i and t_i, the desired capacity on a given edge in the tree is simply the sum of the demands of the commodities whose unique path uses that edge.

Algorithm: Buy-at-Bulk in a tree T

1. Set P_i to be the unique $s_i t_i$-path in T.
2. Set $c_e = \sum_{i : e \in P_i} d_i$ for all $e \in T$.

Let d_{uv} be the length of the shortest path in G using lengths ℓ_e. We use the algorithm to probabilistically approximate d by a tree metric T, then run the algorithm on T. However, since T is over a set of vertices $V' \supseteq V$, we need to translate (V', T) back to the original graph.

Theorem

For any tree metric (V', T) with $V \subseteq V'$ defined by a hierarchical cut decomposition, with vertices in V as the leaves of T, we can find in polynomial time another tree metric (V, T') such that $T_{uv} \leq T'_{uv} \leq 4T_{uv}$.

Proof. Pick any $v \in V$ such that the parent w of v in the tree T is not in V ($w \in V' - V$). We contract edge (v, w), merging the subtree at v into its parent w, and identify the newly merged node as v. Repeat this process until every vertex in the tree is a vertex of V. Finally, multiply the length of every remaining edge by 4. Let T' be the resulting tree.

Clearly $T'_{uv} \leq 4T_{uv}$ since the distance between u and v could have only decreased during contraction of edges, and then increased by a factor of 4 when the edge lengths were multiplied. Now suppose that the least common ancestor of u and v in the original T was a node w at level i so that $T_{uv} = 2^{i+2} - 4$ (from lemma). Then since the contraction process only moves u and v upwards in T, and the nodes for u and v cannot be both merged into w, the distance T'_{uv} in T' must be at least 4 times the length of the edge from w to one of its children, which is $4 \cdot 2^i = 2^{i+2}$. Thus, $T'_{uv} \geq T_{uv}$. ■

From the theorem for tree metrics, we get the following corollary.

Corollary

Given a distance metric (V, d) such that $d_{uv} \geq 1$ for all $u, v \in V, u \neq v$, there is a randomized, polynomial-time algorithm that produces a tree metric (V, T') such that for all $u, v \in V$, $d_{uv} \leq T'_{uv}$ and $E[T'_{uv}] \leq O(\log n) d_{uv}$.

Proof. By proof of the lemma from last section, the least common ancestor of u and v in the tree T obtained from a hierarchical cut decomposition must be on level $\lfloor \log_2 d_{uv} \rfloor$ or higher. By the proof of the previous theorem, the distance $T'_{uv} \geq 2^{i+2}$ for vertices in u and v that have their least common ancestor at level i in tree T. Thus $T'_{uv} \geq d_{uv}$. The other statements follow immediately. ■

Thus, the algorithm uses the algorithm of the corollary to find a tree metric T' and then use the tree buy-at-bulk algorithm on T'. For each edge $(x, y) \in T'$, we find a corresponding shortest path P_{xy} in our input graph G. Then our output path P_i from s_i to t_i in our input metric is the concatenation of the paths P_{xy} for all edges $(x, y) \in T'$ on the $s_i t_i$-path in T'. Given the paths P_i, we set the capacity of e to be the sum of demands routed on paths that use e, so $c_e = \sum_{i:e \in P_i} d_i$.

Algorithm: Buy-at-Bulk Network Design for General Metrics d

1. Apply corollary to find tree metric (V, T') that approximates input metric d.
2. Find shortest path P_{xy} in G for each $(x, y) \in T'$.
3. Let $P'_{s_i t_i}$ be the unique $s_i t_i$-path in T' for all i.
4. Let P_i be the concatenation of paths P_{xy} for all $(x, y) \in P'_{s_i t_i}$ for all i.
5. Set $c_e = \sum_{i:e \in P_i} d_i$ for all $e \in E$.

We show the algorithm is an $O(\log n)$-approximation by relating both the cost of our algorithm's solution and the cost of an optimal solution to the cost of a solution in T'. We let P_{uv} denote the set of edges in a fixed shortest path from u to v in G. Similarly, P'_{uv} is the set of edges in the unique path in T'. Let c'_{xy} be for edges $(x, y) \in T'$, so $c'_{xy} = \sum_{i:(x,y) \in P'_{s_i t_i}} d_i$.

First we show that the cost of the solution can only decrease in the translation from T' to G.

Lemma

The cost of the solution given by the algorithm is at most

$$\sum_{(x,y)\in T'} T'_{xy} f(c'_{xy})$$

Proof. For each $(x, y) \in T'$, our algorithm finds a shortest xy-path in G, P_{xy}. We know that every demand i that uses (x, y) in T' will send its demand in G along this path, so that c'_{xy} demand is sent along this path at a cost of $d_{xy} f(c'_{xy}) \leq T'_{xy} f(c'_{xy})$. It is possible that some edge e in G is contained in more than one shortest path corresponding to edges from T'; for example e might be contained in P_{xy} and P_{vw} corresponding to two edges (x, y) and (v, w) from T'. We will then route demand $c'_{xy} + c'_{vw}$ across e. However, by subadditivity of f, we know that routing multiple paths on e cannot increase the total cost of the solution since $f(c'_{xy} + c'_{vw}) \leq f(c'_{xy}) + f(c'_{vw})$. Thus, we claim that the cost of the solution in T' does not increase when mapped to G.

More precisely,

$$\begin{aligned}
\sum_{(x,y)\in T'} T'_{xy} f(c'_{xy}) &\geq \sum_{(x,y)\in T'} d_{xy} f(c'_{xy}) \\
&= \sum_{(x,y)\in T'} f(c'_{xy}) \sum_{e\in P_{xy}} \ell_e \\
&= \sum_{e\in E} \ell_e \sum_{(x,y)\in T': e\in P_{xy}} f(c'_{xy}) \\
&\geq \sum_{e\in E} \ell_e f\left(\sum_{(x,y)\in T': e\in P_{xy}} c'_{xy} \right) \\
&= \sum_{e\in E} \ell_e f(c_e)
\end{aligned}$$

where the last equality comes from the fact that the total flow that traverses any edge $e \in E$ is exactly the sum of all the c'_{xy} that correspond to $(x, y) \in T$ whose shortest path P_{xy} goes through edge e. ■

Now suppose that an optimal solution uses paths P_i^* in G. Then the optimal solution uses capacity $c_e^* = \sum_{i: e\in P_i^*} d_i$ on edge e and has cost $\text{OPT} = \sum_{e\in E} \ell_e f(c_e^*)$. In order to compare the cost of the optimal solution with the cost on our solution in T', we translate the optimal solution in G to a solution in T'. For each $e = (u, v) \in E$, we install c_e^* units of capacity on all edges in the unique uv-path in T'. The following lemma shows the cost of this translated optimal solution in T' is at least as our solution in T'.

Lemma

The cost of the optimal solution in G translated to T' is at least

$$\sum_{(x,y)\in T'} T'_{xy} f(c'_{xy})$$

Proof. Observe that for any edge $(x, y) \in T'$, our solution uses capacity c'_{xy}, which is exactly equal to the demands of all $s_i t_i$ pairs that would be separated in T' if we removed (x, y) from T'. Any other solution in T' sending d_i units of demand from s_i to t_i for every i must use capacity at least

this much, so the translation of the optimal solution must use capacity at least c'_{xy} on edge (x, y). Thus, since f is nondecreasing and the translation into T' of the optimal solution uses capacity at least c'_{xy} on edge $(x, y) \in T'$ for all $(x, y) \in T'$, the cost of the optimal solution in G translated to T' must be at least $\sum_{(x,y)\in T'} T'_{xy} f(c'_{xy})$. ■

Theorem

The randomized algorithm gives an $O(\log n)$-approximation algorithm for the buy-at-bulk network design problem.

Proof. By combining the two lemmas, we see the cost of the solution given by the algorithm is at most the cost of the optimal solution in G translated to T'. We now only need to show this cost in expectation is at most $O(\log n)\,\mathrm{OPT}$.

Claim: Cost of optimal solution in G translated to T' is $\leq \sum_{e=(u,v)\in E} f(c^*_e) T'_{uv}$.

Proof of Claim. The capacity that the translated solution needs for any edge $(x, y) \in T'$ is $\sum_{e=(u,v)\in E:(x,y)\in P'_{uv}} c^*_e$. By subadditivity, the cost of the optimal solution translated to T' is

$$\begin{aligned}\sum_{(x,y)\in T'} T'_{xy} \cdot f\left(\sum_{e=(u,v)\in E:(x,y)\in P'_{uv}} c^*_e\right) &\leq \sum_{(x,y)\in T'} T'_{xy} \sum_{e=(u,v)\in E:(x,y)\in P'_{uv}} f(c^*_e) \\ &= \sum_{e=(u,v)\in E} f(c^*_e) \sum_{(x,y)\in P'_{uv}} T'_{xy} \\ &= \sum_{e=(u,v)\in E} f(c^*_e) T'_{uv}\end{aligned}$$

■

Given this claim, the theorem follows since the expected cost of the solution is at most

$$E\left[\sum_{e=(u,v)\in E} f(c^*_e) T'_{uv}\right] \leq O(\log n) \sum_{e=(u,v)\in E} f(c^*_e) d_{uv} \leq O(\log n) \sum_{e\in E} f(c^*_e)\ell_e = O(\log n)\,\mathrm{OPT}$$

■

7.3 Linear Arrangement

Problem: Linear Arrangement

Given an undirected graph $G = (V, E)$ and nonnegative weights $w_e \geq 0$ for all $e \in E$, a feasible solution is a one-to-one mapping $f : V \to \{1, \ldots, n\}$, where $n = |V|$. The goal is to find the mapping that minimizes

$$\sum_{e=(u,v)\in E} w_e\,|f(u) - f(v)|$$

Intuitively, this is mapping a graph to points on a line so that we do not stretch the edges by too much. We give an LP relaxation using a metric called a spreading metric. Then we approximate the spreading metric by a tree metric.

The LP relaxation of the linear arrangement problem is

$$\begin{aligned}
\min \quad & \sum_{e=(u,v)\in E} w_e d_{uv} \\
\text{subject to} \quad & \sum_{v\in S} d_{uv} \geq \frac{1}{4}|S|^2, \quad \forall S \subseteq V, u \notin S \\
& d_{uv} = d_{vu}, \quad \forall u, v \in V \\
& d_{uv} \leq d_{uw} + d_{wv}, \quad \forall u, v, w \in V \\
& d_{uv} \geq 1, \quad \forall u, v \in V, u \neq v \\
& d_{uu} = 0, \quad \forall u \in V
\end{aligned}$$

To see this is a relaxation, given a one-to-one mapping $f : V \to \{1, \ldots, n\}$, let d_{uv} be the distance between u and v under the mapping f, i.e. $f_{uv} = |f(u) - f(v)|$. Clearly, the value of the solution is the cost of the linear arrangement.

Also, $d_{uv} = d_{vu}$, $d_{uv} \leq d_{uw} + d_{wv}$, $d_{uv} \geq 1$ if $u \neq v$, and $d_{uu} = 0$. For the final set of constraints, note that for any set of vertices S and any $u \notin S$, there can be at most two vertices in S at distance 1 from $f(u)$ (namely, vertices mapped to $f(u)+1$ and $f(u)-1$ on the line), at most two at distance 2, and so on. Thus, if $|S| = 2k$ or $2k+1$,

$$\sum_{v\in S} d_{uv} \geq 2(1 + \cdots + k) = k(k+1) = \frac{|S|}{2}\left(\frac{|S|}{2} + 1\right) \geq \frac{1}{4}|S|^2$$

where this is the lower bound since if we pack all points close to u, then this is the minimum The variables d arising from the solution to the LP are a metric (V, d) since they obey the three properties of a metric. This metric is called a spreading metric, since the constraint on sets $S \subseteq V$ enforces that for any set S and any $u \notin S$, there is a $v \in S$ that is far from u. Indeed, for any $S \subseteq V$ and any $u \notin S$, let $z = \max_{v\in S} d_{uv}$. Then observe that $z|S| \geq \sum_{v\in S} d_{uv} \geq \frac{1}{4}|S|^2$ implies $z \geq \frac{1}{4}|S|$. The constraint $\sum_{v\in S} d_{uv} \geq \frac{1}{4}|S|^2$ is called the spreading constraint.

Observation

For the spreading metric d, for any subset of vertices S and vertex $u \notin S$, there exist a vertex v such that $d_{uv} \geq \frac{1}{4}|S|$.

The LP can be solved in polynomial time by using the ellipsoid method. The polynomial-time separation oracle for the spreading constraints is as follows: for each $u \in V$, sort the remaining vertices in order of their distance from u, from smallest to largest. Let $v_1, \ldots, v_{n-1}$ such that $d_{uv_1} \leq \cdots \leq d_{uv_{n-1}}$. We then check the constraint for each of the sets $\{v_1\}, \ldots, \{v_1, v_2\}, \ldots, \{v_1, \ldots, v_{n-1}\}$. We claim that if the constraint is not violated for any of these sets for any u, then no constraint is violated. Suppose the constraint is violated for some S and $u \notin S$. Then clearly the sum $\sum_{v\in S} d_{uv}$ is at least as large as $\sum_{i=1}^{|S|} d_{uv_i}$, so if the constraint is violated for S and $u \notin S$, it will also be violated for u and the set $\{v_1, \ldots, v_{|S|}\}$.

To get an approximation algorithm, we use a tree metric to approximate d_{uv} obtained by solving the LP. Here, we can obtain a tree metric deterministically. In particular, we show that for any metric d and any set of nonnegative costs c_{uv} on $u, v \in V$, one can find in polynomial time a tree metric (V', T) such that $T_{uv} \geq d_{uv}$ and

$$\sum_{u,v\in V} c_{uv} T_{uv} \leq O(\log n) \sum_{u,v\in V} c_{uv} d_{uv}$$

The randomized tree metric before gets us

$$E\left[\sum_{u,v\in V} c_{uv}T_{uv}\right] \leq O(\log n)\sum_{u,v\in V} c_{uv}d_{uv}$$

We need the tree metric T to have additional properties: first, we need that T is a rooted tree with all the vertices of V at the leaves of the tree, and second, if vertex $z \in V$ belongs to the smallest subtree containing vertices $u, v \in V$, then $T_{uv} \geq T_{uz}$. These additional properties are satisfied by trees resulting from a hierarchical cut decomposition. We can find these trees efficiently (by Theorem 7.3).

Theorem

Given a metric d_{uv} on vertices V and nonnegative costs c_{uv} for all $u, v \in V$, in polynomial time we can compute a tree metric (V', T) on a set of vertices $V' \supseteq V$ such that $T_{uv} \geq d_{uv}$ for all $u, v \in V$ and $\sum_{u,v\in V} c_{uv}T_{uv} \leq O(\log n)\sum_{u,v\in V} c_{uv}d_{uv}$.
Furthermore, T is a rooted tree with all vertices of V at its leaves, and if vertex $z \in V$ belongs to the smallest subtree containing vertices $u, v \in V$, then $T_{uv} \geq T_{uz}$.

Algorithm: Linear Arrangement

1. Solve the LP relaxation to obtain metric d, use Theorem 7.3 to obtain tree metric (V', T) with costs $c_{uv} = w_e$ for each $e = (u, v) \in E$ and $c_{uv} = 0$ otherwise.
2. Assign each leaf of the tree T a number from 1 to n (intuitively, number them consecutively left to right).

Theorem

The algorithm is an $O(\log n)$-approximation algorithm for the linear arrangement problem.

Proof. Let f be the one-to-one mapping produced by the algorithm. For any given edge $e = (u, v)$, consider the smallest subtree in the tree T that contains both u and v. This subtree is assigned some range of integers $[a, b]$. Since the leaves of the subtree are numbered consecutively, there are $b - a + 1$ leaves in the subtree. At worst, we can have one endpoint of the edge assigned to a and the other to b, hence $|f(u) - f(v)| \leq b - a$.

Let S be the set of leaves in the subtree other than u, we have $|S| = b - a$. By the observation, we know there is some other vertex $z \in S$ such that $d_{uz} \geq \frac{1}{4}(b - a)$. Therefore, by the property of the tree metric,

$$T_{uv} \geq T_{uz} \geq d_{uz} \geq \frac{1}{4}(b - a)$$

since z belongs to the smallest subtree containing u and v. Hence, we have $|f(u) - f(v)| \leq 4T_{uv}$. By the previous theorem, we can bound the cost of f

$$\sum_{e=(u,v)\in E} w_e\,|f(u) - f(v)| \leq 4\sum_{e=(u,v)\in E} w_eT_{uv} \leq O(\log n)\sum_{e=(u,v)\in E} w_ed_{uv} \leq O(\log n)\,\mathrm{OPT}$$

where final inequality follows since $\sum_{e=(u,v)\in E} w_ed_{uv}$ is the objective function of the LP. ■

To begin the proof of Theorem 7.3, we will need to partition S corresponding to a node at level i to obtain its children at level $i - 1$. As before, we define the concept of a ball and volume of the

ball. For a given metric d, let $B_d(u,r) = \{v \in V : d_{uv} \le r\}$. Let $V^* = \sum_{u,v \in V} c_{uv} d_{uv}$. We define the volume $V_d(u,r)$ as

$$V_d(u,r) = \frac{V^*}{n} + \sum_{v,w \in B_d(u,r)} c_{vw} d_{vw} + \sum_{v \in B_d(u,r), w \notin B_d(u,r)} c_{vw}(r - d_{uv})$$

Also, define $c(\delta(B_d(u,r)))$ to be the cost of the pairs of vertices with exactly one vertex in $B_d(u,r)$ (shorthand $c_d(u,r) = c(\delta(B_d(u,r)))$), so that

$$c_d(u,r) = c(\delta(B_d(u,r))) = \sum_{v \in B_d(u,r), w \notin B_d(u,r)} c_{vw}$$

Here, instead of making a random choice for the radii, we can make good, deterministic choices via the region growing technique that relates cost $c_d(u,r)$ to the volume of the ball.

Lemma

In polynomial time, it is possible to find a value r for any $u \in V$ and $i, 0 \le i \le \log_2 \Delta$ such that $2^{i-1} \le r < 2^i$ and

$$c_d(u,r) \le 2^{1-i} \ln \left(\frac{V_d(u,2^i)}{V_d(u,2^{i-1})} \right) V_d(u,r)$$

Proof. This follows from the corollary

Corollary

Given lengths x_e on edges $e \in E$ and a vertex u, one can find in polynomial time a radius $r \in [a,b)$ such that

$$c(\delta(B_x(u,r))) \le \frac{1}{b-a} \ln \left(\frac{V_x(u,b)}{V_x(u,a)} \right) V_x(u,r)$$

Apply this corollary with interval $[2^{i-1}, 2^i)$. ■

The partitioning of a node corresponding to S at level i into children at level $i-1$ is performed as follows: instead of using a random ordering of vertices, find a vertex $u \in S$ that maximizes the volume $V_d(u, 2^{i-2})$ and find a ball of radius $r, 2^{i-2} \le r < 2^{i-1}$ around u via the Lemma. Make a child node of S corresponding to all vertices of $B_d(u,r)$ in S, remove these from S and repeat until $S = \emptyset$. The ball has radius less than 2^{i-1} as required.

Proof of Theorem 7.3. The distance between u and v depends on the lowest common ancestor of u and v. Recall $T_{uv} = 2^{i+2} - 4$. Thus, if z is in the smallest subtree containing u and v, clearly the least common ancestor of u and z is at level at most i, so $T_{uv} \ge T_{uz}$.

Now we show the $O(\log n)$ approximation. For a given pair $u, v \in V$, let $i+1$ be the level of the least common ancestor in T containing both u and v. We showed that $T_{uv} \le 2^{i+3}$. Let E_{i+1} be the pairs of vertices (u,v) with $u, v \in V$ whose least common ancestor in T at level $i+1$. Thus,

$$\sum_{u,v \in V} c_{uv} T_{uv} \le \sum_{i=0}^{\log_2 \Delta - 1} \sum_{(u,v) \in E_{i+1}} 2^{i+3} \cdot c_{uv}$$

Note that $(u,v) \in E_{i+1}$ implies there is a node at level i in T corresponding to a ball such that exactly one of u and v is inside the ball. Thus, $\sum_{(u,v) \in E_{i+1}} c_{uv}$ is at most the sum of the cuts

created when we formed the children at level i and by the previous lemmas, we can relate these cuts to the volume of these children.

Let C_i be the set of centers of balls of nodes at level i, and for $z \in C_i$, let r_{zi} be the radius of the ball around z we chose via the lemma. Then for any level i, $\sum_{(u,v)\in E_{i+1}} c_{uv} \leq \sum_{z\in C_i} c_d(z, r_{zi})$. By lemma, we know $c_d(z, r_{zi}) \leq 2^{1-i} \ln\left(\frac{V_d(z,2^i)}{V_d(z,2^{i-1})}\right) V_d(z, r_{zi})$. Thus,

$$\begin{aligned}
\sum_{u,v\in V} c_{uv}T_{uv} &\leq \sum_{i=0}^{\log_2\Delta-1} \sum_{(u,v)\in E_{i+1}} 2^{i+3}\cdot c_{uv} \\
&\leq \sum_{i=0}^{\log_2\Delta-1} \sum_{z\in C_i} 2^{i+3}\cdot c_d(z, r_i) \\
&\leq 16 \sum_{i=0}^{\log_2\Delta-1} \sum_{z\in C_i} \ln\left(\frac{V_d(z,2^i)}{V_d(z,2^{i-1})}\right) V_d(z, r_{zi})
\end{aligned}$$

To relate the final term to the overall volume, let $g(v)$ be the volume of all the edges incident to v plus V^*/n, that is $g(v) = \frac{V^*}{n} + \sum_{u\in V} c_{uv}d_{uv}$. Then the volume of the set associated with any node in the tree is $\leq$ sum of $g(v)$ for all v in the set, that is $V_d(z, r_{zi}) \leq \sum_{v\in S} g(v)$.

Pick any $v \in S$, since $r_{zi} < 2^i$, any edge or part of an edge contributing volume to the ball of radius r_{zi} in S must also be in a ball of radius 2^{i+1} around v. Thus, $V_d(z, r_{zi}) \leq V_d(v, 2^{i+1})$.

By construction, if z is a center of a ball at level i, it must be that $V_d(z, 2^{i-1}) \geq V_d(v, 2^{i-1})$ for any other $v \in S$ since we chose $z \in S$ to maximize the volume $V_d(z, 2^{i-1})$. Thus,

$$\ln\left(\frac{V_d(z,2^i)}{V_d(z,2^{i-1})}\right) V_d(z, r_{zi}) \leq \sum_{v\in S} \ln\left(\frac{V_d(z,2^i)}{V_d(z,2^{i-1})}\right) g(v) \leq \sum_{v\in S} \ln\left(\frac{V_d(v,2^{i+1})}{V_d(v,2^{i-1})}\right) g(v)$$

Substituting back into the original inequality,

$$\begin{aligned}
\sum_{u,v\in V} c_{uv}T_{uv} &\leq 16 \sum_{i=0}^{\log_2\Delta-1} \sum_{v\in V} \ln\left(\frac{V_d(v,2^{i+1})}{V_d(v,2^{i-1})}\right) g(v) \\
&\leq 16 \sum_{v\in V} \sum_{i=0}^{\log_2\Delta-1} \ln\left(\frac{V_d(v,2^{i+1})}{V_d(v,2^{i-1})}\right) g(v)
\end{aligned}$$

For each $v \in V$, the sum telescopes to $\ln(V_d(v,\Delta)) + \ln(V_d(v,\Delta/2)) - \ln(V_d(v,1)) - \ln(V_d(v,1/2))$, which can be further bounded above by $2(\ln(V_d(v,\Delta)) - \ln(V_d(v,0)))$. So the sum is at most

$$32 \sum_{v\in V} \frac{V_d(v,\Delta)}{V_d(v,0)} g(v)$$

Now since $V_d(v,\Delta) \leq V^* + \frac{V^*}{n}$ and $V_d(v,0) = V^*/n$,

$$32 \sum_{v\in V} \frac{V_d(v,\Delta)}{V_d(v,0)} g(v) \leq 32 \sum_{v\in V} \frac{V^* + V^*/n}{V^*/n} g(v) = 32\ln(n+1) \sum_{v\in V} g(v)$$

By definition of $g(v)$,

$$\begin{aligned}\sum_{u,v\in V} c_{uv}T_{uv} &\leq 32\ln(n+1)\sum_{v\in V} g(v)\\ &= 32\ln(n+1)\sum_{v\in V}\left(\frac{V^*}{n}+\sum_{u\in V} c_{uv}d_{uv}\right)\\ &= 96\ln(n+1)\sum_{u,v\in V} c_{uv}d_{uv}\\ &\leq O(\log n)\sum_{u,v\in V} c_{uv}d_{uv}\end{aligned}$$

■

7.4 Sparsest Cut

We can approximate a general metric with another metric more general than a tree metric, the ℓ_1-embeddable metric. Recall the ℓ_1 norm,

$$\|x\|_1 = \sum_{i=1}^{m} x^i$$

where x^i is the ith component.

Definition: ℓ_1-Embeddable Metric

A metric (V,d) if there exists a function $f : V \to \mathbb{R}^m$ for some m such that $d_{uv} = \|f(u)-f(v)\|_1$ for all $u,v \in V$.

The function f is called the embedding of the metric into ℓ_1. Any tree metric is an ℓ_1-embeddable metric.

The ℓ_1-embeddable metrics are closely related to cuts. We will show that any ℓ_1 metric (V,d) is a weighted sum of cuts of V. Formally, let $\chi_{\delta(S)}(u,v)$ be an indicator function for whether edge (u,v) is in $\delta(S)$.

Lemma

Let (V,d) be an ℓ_1-embeddable metric, and let $f : V \to \mathbb{R}^m$ be the associated embedding. Then there exist $\lambda_S \geq 0$ for all $S \subseteq V$ such that for all $u,v \in V$,

$$\|f(u)-f(v)\|_1 = \sum_{S\subseteq V} \lambda_S \chi_{\delta(S)}(u,v)$$

Furthermore, at most mn of the λ_S are nonzero, and the λ_S can be computed in polynomial time.

Definition: ℓ_1 Distortion

A metric (V, d) embeds into ℓ_1 with distortion α if there exists a function $f : V \to \mathbb{R}^m$ for some m and r such that

$$r \cdot d_{uv} \leq \|f(u) - f(v)\|_1 \leq r\alpha \cdot d_{uv}$$

for all $u, v \in V$.

Theorem

Any metric (V, d) embeds into ℓ_1 with distortion $O(\log n)$. Furthermore, the embedding $f : V \to \mathbb{R}^{(\log^2 n)}$ is computable with high probability in randomized polynomial-time.

Problem: Sparsest Cut

Given an undirected graph $G = (V, E)$, costs $c_e \geq 0$ for all $e \in E$, and k pairs of vertices s_i, t_i with associated positive integer demands d_i, find a set of vertices S that minimizes

$$\rho(S) = \frac{\sum_{e \in \delta(S)} c_e}{\sum_{i:|S \cap \{s_i, t_i\}|=1} d_i}$$

the ratio of the cost of edges in the cut to the sum of the demands separated by the cut.

The LP relaxation is

$$\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e \\
\text{subject to} \quad & \sum_{i=1}^{k} d_i y_i = 1 \\
& \sum_{e \in P} x_e \geq y_i, \quad \forall P \in \mathcal{P}_i, 1 \leq i \leq k \\
& y_i \geq 0, \quad 1 \leq i \leq k \\
& x_e \geq 0, \quad \forall e \in E
\end{aligned}$$

Theorem

There is a randomized $O(\log n)$-approximation algorithm for the sparsest cut problem.

Theorem

Given a metric (V, d) and k pairs of distinguished vertices, $s_i, t_i \in V, 1 \leq i \leq k$, we can in randomized polynomial-time compute an embedding $f : V \to \mathbb{R}^{O(\log^2 k)}$ such that with high probability, $\|f(u) - f(v)\|_1 \leq r \cdot O(\log k) \cdot d_{uv}$ for all $u, v \in V$ and $\|f(s_i) - f(t_i)\|_1 \geq r \cdot d_{s_i t_i}$ for $1 \leq i \leq k$.

Corollary

There is a randomized $O(\log k)$-approximation for the sparsest cut problem.

This uses the Fréchet embedding which maps a metric (V, d) to points in Euclidean space. Given (V, d) and a set $A \subseteq V$, define $d(u, A) = \min_{v \in A} d_{uv}$ for every $u \in V$.

Definition: Fréchet Embedding

Given a metric space (V, d) and p subsets of vertices $A_1, \ldots, A_p$, a Fréchet embedding $f : V \to \mathbb{R}^p$ is defined by

$$f(u) = (d(u, A_1), d(u, A_2), \ldots, d(u, A_p)) \in \mathbb{R}^p$$

for all $u \in V$.

The embedding is easy to compute given the sets A_i. In Euclidean space, the ℓ_1 distance between points is at most the dimension p times the original distance.

Lemma

Given a metric (V, d) and the Fréchet embedding $f : V \to \mathbb{R}^p$, for any $u, v \in V$, $\|f(u) - f(v)\|_1 \leq p \cdot d_{uv}$.

Lemma

Given a metric space (V, d) with k distinguished pairs $s_i, t_i \in V$, we can pick $p = O(\log^2 k)$ sets $A_j \subseteq V$ using randomization such that a Fréchet embedding $f : V \to \mathbb{R}^p$ gives $\|f(s_i) - f(t_i)\|_1 \geq \Omega(\log k) \cdot d_{s_i t_i}$ for $1 \leq i \leq k$ with high probability.

Given the embedding and lemmas, we can use them to prove the $O(\log k)$-approximation for sparsest cut.

Assuming unique games conjecture, there is no α-approximation algorithm for the sparsest cut problem for constant α, unless $\mathbf{P} = \mathbf{NP}$.

Chapter 8

Semidefinite Programming

We study another class of relaxations, called vector programs. Vector programs are equivalent to a powerful and well-studied generalization of linear programs, called semidefinite programs. SDPs and consequently VPs can be solved within an additive error of $\varepsilon > 0$, in time polynomial in n and $\log(1/\varepsilon)$ using the ellipsoid method.

8.1 Quadratic Programs and Vector Programs

Definition: Quadratic Program

The problem of optimizing a quadratic function of integer valued variables, subject to quadratic constraints on these variables.

Definition: Strict Quadratic Program

Each monomial in the objective function and in each of the constraints is of degree 0 (constant) or 2.

Problem: Maximum Cut (MAX-CUT)

Given an undirected graph $G = (V, E)$ with edge weights $w : E \to \mathbb{Q}^+$, find a partition $(S, \overline{S})$ of V so as to maximize the total weight of edges in this cut.

We now give a strict quadratic program for MAX-CUT. Let $y_i \in \{-1, +1\}$ be an indicator variable for vertex v_i. The partition $(S, \overline{S})$ will be $S = \{v_i : y_i = 1\}$ and $\overline{S} = \{v_i : y_i = -1\}$.

If v_i and v_j are on opposite sides of this partition, $y_i y_j = -1$ and edge (v_i, v_j) contributes w_{ij} to the objective. $y_i y_j = 1$ if they are on the same side and the edge makes no contribution. Hence, an optimal solution to this program is a max cut in G.

$$\begin{aligned} \max \quad & \frac{1}{2} \sum_{1 \le i < j \le n} w_{ij}(1 - y_i y_j) \\ \text{subject to} \quad & y_i^2 = 1, \quad \forall v_i \in V \\ & y_i \in \mathbb{Z}, \quad \forall v_i \in V \end{aligned} \tag{MAX-CUT QP}$$

Definition: Vector Program

Defined over n vector variables $v_1, \ldots, v_n$ in $\mathbb{R}^n$ and is the problem of optimizing a linear function of the inner products $v_i \cdot v_j, 1 \leq i \leq j \leq n$ subject to linear constraints on these inner products.

A VP can be thought of as being obtained from an LP by replacing each variable with an inner product of a pair of these vectors.

We will relax the QP to a VP.

$$\begin{aligned} \max \quad & \frac{1}{2} \sum_{1 \leq i < j \leq n} w_{ij}(1 - v_i \cdot v_j) \\ \text{subject to} \quad & v_i \cdot v_i = 1, \quad \forall v_i \in V \\ & v_i \in \mathbb{R}^n, \quad \forall v \in V \end{aligned} \qquad \textbf{(MAX-CUT VP)}$$

Because of the constraint $v_i \cdot v_i = 1$, the vectors are constrained to lie in the n-dimensional sphere S_{n-1}. Any feasible solution to **MAX-CUT QP** yields a solution to **MAX-CUT VP** having the same objective value, by assigning the vector $(y_i, 0, \ldots, 0)$ to v_i. Therefore, the vector program is a relaxation of the strict quadratic program.

Vector programs are approximable to any desired degree of accuracy in polynomial-time, so the relaxation provides an upper bound on OPT for MAX-CUT. Vector programs do not always come as relaxations of strict quadratic programs.

8.2 Properties of Positive Semidefinite Matrices

Let A be a real, symmetric $n \times n$ matrix. A has real eigenvalues and has n linearly independent eigenvectors.

Definition: Positive Semidefinite Matrix

Let A be a real, symmetric $n \times n$ matrix. A has real eigenvalues and has n linearly independent eigenvectors. We say A is PSD if

$$\forall x \in \mathbb{R}^n, x^T A x \geq 0$$

Theorem

Let A be a real, symmetric $n \times n$ matrix. Then the following are equivalent:

1. $\forall x \in \mathbb{R}^n, x^T A x \geq 0$, i.e. A is PSD.
2. All eigenvalues of A are nonnegative.
3. There is an $n \times n$ real matrix W such that $A = W^T W$.

Proof. $(1 \implies 2)$: Let λ be an eigenvalue of A and let v be a corresponding eigenvector. Therefore, $Av = \lambda v$. Pre-multiplying by v^T, we get $v^T A v = \lambda v^T v$. By 1, $v^T A v \geq 0$, so $\lambda v^T v \geq 0$. Since $v^T v > 0$, $\lambda \geq 0$.

(2 $\implies$ 3): Let $\lambda_1, \ldots, \lambda_n$ be n eigenvalues of A and $v_1, \ldots, v_n$ be the corresponding complete collection of orthonormal eigenvectors. Let Q be the matrix whose columns are $v_1, \ldots, v_n$ and Λ be the diagonal matrix with entries $\lambda_1, \ldots, \lambda_n$. Since for each i, $Av_i = \lambda_i v_i$, we have $AQ = Q\Lambda$. Since Q is orthogonal, i.e. $QQ^T = I$, we get that $Q^T = Q^{-1}$. Therefore,

$$A = Q\Lambda Q^T$$

Let D be the diagonal matrix whose diagonal entries are the positive square roots of $\lambda_1, \ldots, \lambda_n$ (2 gives the eigenvalues as nonnegative, so square roots are real). Then $\Lambda = DD^T$. Substituting,w e get

$$A = QDD^TQ^T = (QD)(QD)^T$$

The result follows by letting $W = (QD)^T$.

(3 $\implies$ 1): For any $x \in \mathbb{R}^n$, $x^TAx = x^TW^TWx = (Wx)^T(Wx) \geq 0$. ■

Using Cholesky decomposition, a real symmetric matrix can be decomposed in polynomial time as $A = U\Lambda U^T$, where Λ is the diagonal matrix with eigenvalues as entries. It is also easy to see that any convex combination of PSD matrices is PSD.

8.3 Semidefinite Programming

Definition: Semidefinite Program

Let Y be an $n \times n$ matrix of real valued variables with entries y_{ij}. The problem of optimizing a linear function of the y_{ij}'s, subject to linear constraints on them and the additional constraint that Y is symmetric and PSD.

We formalize this by denoting $\mathbb{R}^{n\times n}$ as the space of $n \times n$ real matrices, and the Frobenius inner product of $A, B \in \mathbb{R}^{n\times n}$ is

$$A \bullet B = tr(A^TB) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}b_{ij}$$

Let M_n denote the cone of symmetric $n \times n$ real matrices. For $A \in M_n$, $A \succeq 0$ denotes the fact that A is PSD.

Let $C, D_1, \ldots, D_k \in M_n$ and $d_1, \ldots, d_k \in \mathbb{R}$. The general SDP problem $\mathcal{S}$ is

$$\begin{aligned} \max \quad & C \bullet Y \\ \text{subject to} \quad & D_i \bullet Y = d_i, \quad 1 \leq i \leq k \\ & Y \succeq 0 \\ & Y \in M_n \end{aligned}$$

Observe that if $C, D_1, \ldots, D_k$ are all diagonal matrices, this is simply an LP.

The set of feasible solutions is convex for an SDP. Let $A \in \mathbb{R}^{n\times n}$ be an infeasible point. Let $C \in \mathbb{R}^{n\times n}$. A hyperplane $C \bullet Y \leq b$ is called a separating hyperplane for A if all feasible points satisfy it and point A does not.

Theorem

Let $\mathcal{S}$ be a semidefinite programming problem, and A be a point in $\mathbb{R}^{n\times n}$. We can determine, in polynomial time, whether A is feasible for $\mathcal{S}$, and if it is not, find a separating hyperplane.

Proof. Testing for feasibility involves ensuring A is symmetric and PSD and that it satisfies all linear constraints. This can be done in polynomial time.

If A is infeasible, a separating hyperplane is obtained as follows:

- If A is not symmetric, $a_{ij} > a_{ji}$ for some i, j, then $y_{ij} \leq y_{ji}$ is a separating hyperplane.
- If A is not PSD, then it has a negative eigenvalue λ. Let v be the corresponding eigenvector. Now $(vv^T) \bullet Y = v^T Y v \geq 0$ is a separating hyperplane.
- If any of the linear constraints is violated, it directly yields a separating hyperplane.

■

Let $\mathcal{V}$ be a vector program on n n-dimensional vector variables $v_1, \ldots, v_n$. Define the corresponding semidefinite program $\mathcal{S}$ over n^2 variables $y_{ij}, 1 \leq i, j \leq n$ by replacing each inner product $v_i \cdot v_j$ occurring in $\mathcal{V}$ by the variable y_{ij}. The objective function and constraints are now linear in y_{ij}'s. Additionally, require that matrix Y, whose (i, j)th entry is y_{ij} is symmetric and PSD.

Lemma

Vector program $\mathcal{V}$ is equivalent to semidefinite program $\mathcal{S}$.

Proof. We show that for each feasible solution to $\mathcal{V}$, there is a feasible solution to $\mathcal{S}$ with the same objective value and vice versa. Let $v_1, \ldots, v_n$ be a feasible solution to $\mathcal{V}$. Let W be the matrix whose columns are $v_1, \ldots, v_n$. Then $A = W^T W$ is a feasible solution to $\mathcal{S}$ having the same objective value.

Let A be a feasible solution to $\mathcal{S}$. By the Theorem from last section on PSD equivalency, there is an $n \times n$ matrix W such that $A = W^T W$. Let $v_1, \ldots, v_n$ be the columns of W. Then $v_1, \ldots, v_n$ is a feasible solution to $\mathcal{V}$ having the same objective value. ■

8.4 Maximum Cut

The semidefinite programming relaxation to MAX-CUT that is equivalent to the vector program **MAX-CUT VP** is

$$\begin{aligned} \max \quad & \frac{1}{2} \sum_{1 \leq i < j \leq n} w_{ij}(1 - y_i y_j) && \textbf{(MAX-CUT SDP)} \\ \text{subject to} \quad & y_i^2 = 1, \quad \forall v_i \in V \\ & Y \succeq 0 \\ & Y \in M_n \end{aligned}$$

Assume that we have an optimal solution to the VP for MAX-CUT. The slight inaccuracy can be absorbed into the approximation factor. Let $v_1, \ldots, v_n$ be an optimal solution and let OPT_{SDP} denote its objective value. We need obtain a cut $(S, \overline{S})$ whose weight is a large fraction of OPT_{SDP}.

Let θ_{ij} denote the angle between vectors v_i and v_j. The contribution of this pair of vectors to OPT_{SDP} is

$$\frac{w_{ij}}{2}(1 - \cos\theta_{ij})$$

since $v_i \cdot v_j = \|v_i\| \, \|v_j\| \cos(\theta_{ij}) = 1 \cdot 1 \cdot \cos(\theta_{ij})$. The closer θ_{ij} is to π, the larger this contribution. So we want i and j to be separated if θ_{ij} is large. We do this by picking r to be a uniformly distributed vector on the unit sphere S_{n-1} and let $S = \{i \in V : v_i \cdot r \geq 0\}$.

Lemma

$$\Pr[i \text{ and } j \text{ are separated}] = \frac{\theta_{ij}}{\pi} = \frac{\arccos(v_i \cdot v_j)}{\pi}$$

Proof. Project r onto the plane containing v_i and v_j. Vertices v_i and v_j will be separated if and only if the projection lies in one of the two arcs of angle θ_{ij}. Since r is picked from a spherically symmetric distribution, its projection will be a random direction on this plane.

The actual probability comes from the hyperplane normal to r, since we define S as being $v_i \cdot r \geq 0$, i.e. 90° or less, and if the dot product is < 0, then we have an angle larger than 90°. We want this normal to r to lie in between the v_i and v_j. A picture is shown below. It is equivalent to choosing an r that lies in between. ■

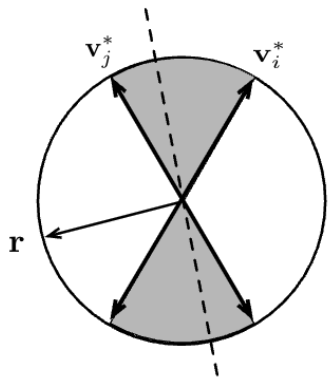


Figure 8.1: A vector r such that $v_j^* \cdot r > 0$ and $v_i^* \cdot r < 0$. The dashed line is the normal vector to r, which passes between v_i^* and v_j^*.

Lemma

Let $x_1, \ldots, x_n$ be picked independently from the normal distribution with mean 0 and standard deviation 1. Let $d = \sqrt{x_1^2 + \cdots + x_n^2}$. Then $(x_1/d, \ldots, x_n/d)$ is a random vector on the unit sphere S_{n-1}.

Proof. Consider the vector $r = (x_1, \ldots, x_n)$. The distribution function for r has density

$$f(y_1, \ldots, y_n) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} e^{-y_i^2/2} = \frac{1}{(2\pi)^{n/2}} e^{-\frac{1}{2}\sum_i y_i^2}$$

The density function depends only on the distance of the point from the origin. Therefore, the distribution of r is spherically symmetric. Hence, dividing by the length of r, d, we get a random vector in S_{n-1}. ■

Algorithm: 0.87856-Approximation for Maximum Cut

1. Solve vector program **MAX-CUT VP** for optimal solution $v_1, \ldots, v_n$.
2. Pick r to be a uniformly distributed vector on unit sphere S_{n-1}.
3. $S = \{i \in V : v_i \cdot r \geq 0\}$.

Lemma

For $x \in [-1, 1]$ and $\alpha > 0.87856$,

$$\frac{1}{\pi} \arccos x \geq 0.87856 \cdot \frac{1}{2}(1 - x)$$

Thus, $\alpha = \frac{2}{\pi} \min_{0 \leq \theta \leq \pi} \frac{\theta}{1 - \cos\theta} > 0.87856$.

Theorem

The randomized hyperplane rounding for **MAX-CUT VP** is a 0.878-approximation algorithm for maximum cut.

Proof. Let X_{ij} be the random variable for edge (i, j) being in the cut or not. Let W be the random variable which gives the weight of the cut. We have $W = \sum_{(i,j) \in E} w_{ij} X_{ij}$. Then by the probability lemma,

$$E[W] = \sum_{(i,j) \in E} w_{ij} \cdot \Pr[(i, j) \in \delta(S)] = \sum_{(i,j) \in E} w_{ij} \cdot \frac{\theta}{\pi} = \sum_{(i,j) \in E} w_{ij} \cdot \frac{1}{\pi} \arccos(v_i \cdot v_j)$$

We can bound this by the lemma above

$$E[W] \geq 0.878 \cdot \frac{1}{2} \sum_{(i,j) \in E} w_{ij}(1 - v_i \cdot v_j) = 0.878 \cdot \mathrm{OPT}_{VP/SDP} \geq 0.878 \cdot \mathrm{OPT}$$

■

Theorem

If there is an α-approximation algorithm for maximum cut with $\alpha > \frac{16}{17} \approx 0.941$, then $\mathbf{P} = \mathbf{NP}$.

Theorem

Given the unique games conjecture, there is no α-approximation algorithm for maximum cut with constant

$$\alpha > \min_{-1 \leq x \leq 1} \frac{\arccos(x)/\pi}{\frac{1}{2}(1 - x)} \geq 0.878$$

unless $\mathbf{P} = \mathbf{NP}$.

8.5 MAX-2SAT

Recall the $\frac{3}{4}$-approximation for MAX-SAT using the better of two randomized rounding algorithms. We can give an improved approximation for the restriction in which each clause contains at most

two literals.

We formulate a strict quadratic program. Corresponding to each Boolean variable x_i, we have $y_i \in \{-1, +1\}$ for $1 \leq i \leq n$. In addition, we introduce $y_0 \in \{-1, +1\}$. We impose that x_i is true if $y_i = y_0$ and false otherwise. We also write $v(C)$ as the value of clause C which takes 1 if C is satisfied and 0 otherwise.

For clauses containing one literal,

$$v(x_i) = \frac{1 + y_0 y_i}{2}, v(\overline{x}_i) = \frac{1 - y_0 y_i}{2}$$

For a clause with 2 literals,

$$\begin{aligned} v(x_i \vee x_j) &= 1 - v(\overline{x}_i)v(\overline{x}_j) \\ &= 1 - \frac{1 - y_0 y_i}{2}\frac{1 - y_0 y_j}{2} \\ &= \frac{1}{4}(3 + y_0 y_i + y_0 y_j - y_0^2 y_i y_j) \\ &= \frac{1 + y_0 y_i}{4} + \frac{1 + y_0 y_j}{4} + \frac{1 - y_i y_i}{4} \end{aligned}$$

We used the fact that $y_0^2 = 1$. A 2 literal clause consists of a linear combination of terms of $(1+y_i y_j)$ or $(1 - y_i y_j)$. The strict quadratic program is

$$\begin{aligned} \max \quad & \sum_{0 \leq i < j \leq n} a_{ij}(1 + y_i y_j) + b_{ij}(1 - y_i y_j) \\ \text{subject to} \quad & y_i^2 = 1, \quad 0 \leq i \leq n \\ & y_i \in \mathbb{Z}, \quad 0 \leq i \leq n \end{aligned}$$

The vector program relaxation is

$$\begin{aligned} \max \quad & \sum_{0 \leq i < j \leq n} a_{ij}(1 + v_i \cdot v_j) + b_{ij}(1 - v_i \cdot v_j) \\ \text{subject to} \quad & v_i \cdot v_i = 1, \quad 0 \leq i \leq n \\ & v_i \in \mathbb{R}^{n+1}, \quad 0 \leq i \leq n \end{aligned}$$

The algorithm is similar to MAX-CUT.

Algorithm: 0.878-Approximation for MAX-2SAT

1. Solve the VP for $v_0, \ldots, v_n$.
2. Pick r uniformly distributed on S_n, the unit sphere in $n+1$-dimensions.
3. $y_i = 1$ if and only if $r \cdot v_i \geq 0$ for $0 \leq i \leq n$.

Theorem

$$E[W] \geq \alpha \cdot \text{OPT}$$

Proof.

$$E[W] = 2 \sum_{0 \leq i < j \leq n} a_{ij} \Pr[y_i = y_j] + b_{ij} \Pr[y_i \neq y_j]$$

Let θ_{ij} denote the angle between v_i and v_j. Then

$$\Pr[y_i \neq y_j] = \frac{\theta_{ij}}{\pi} \geq \frac{\alpha}{2}(1 - \cos\theta_{ij})$$

and

$$\Pr[y_i = y_j] = 1 - \frac{\theta_{ij}}{\pi} \geq \frac{\alpha}{2}(1 + \cos\theta_{ij})$$

Therefore,

$$E[W] \geq \alpha \sum_{0 \leq i < j \leq n} a_{ij}(1 + \cos\theta_{ij}) + b_{ij}(1 - \cos\theta_{ij}) = \alpha \cdot \text{OPT}$$

■

8.6 Approximating Quadratic Programs

We can extend the algorithm for MAX-CUT to a more general problem. Suppose we want to approximate the QP

$$\begin{aligned} \max \quad & \sum_{1 \leq i,j \leq n} a_{ij} x_i x_j \\ \text{subject to} \quad & x_i \in \{-1, +1\}, \quad \forall i = 1, \ldots, n \end{aligned} \tag{QP}$$

We need to be more careful since the value of OPT can be negative. To get around this, we restrict the objective function matrix $A = (a_{ij})$ to be itself PSD. Then any feasible solution x will have $x^T Ax$ be nonnegative by definition.

We get the VP relaxation

$$\begin{aligned} \max \quad & \sum_{1 \leq i,j \leq n} a_{ij}(v_i \cdot v_j) \\ \text{subject to} \quad & v_i \cdot v_i = 1, \quad \forall i = 1, \ldots, n \\ & v_i \in \mathbb{R}^n, \quad \forall i = 1, \ldots, n \end{aligned}$$

In the same way, $\text{OPT}_{VP} \geq \text{OPT}$. We can also use the same algorithm as MAX-CUT. Solve the VP to get v_i, choose a random hyperplane with normal r, and generate a solution x for the QP by setting $x_i = 1$ if $r \cdot v_i \geq 0$ and $x_i = -1$ otherwise.

Lemma

$$E[x_i x_j] = \frac{2}{\pi} \arcsin(v_i \cdot v_j)$$

Theorem $\frac{2}{\pi}$-Approximation for Quadratic Programs

The vector program random hyperplane rounding algorithm is a $\frac{2}{\pi}$-approximation algorithm for the quadratic program **QP** when the objective function matrix A is positive semidefinite.

8.7 Coloring 3-Colorable Graphs

Definition: $\tilde{O}$

A function $g(n) = \tilde{O}(f(n))$ if there exists some constant $c \geq 0$ and some n_0 such that for all $n \geq n_0$, $g(n) = O(f(n)\log^c n)$.

We first color a 3-colorable graph G with $O(\sqrt{n})$ colors. Then by SDPs, we can obtain an algorithm that uses $\tilde{O}(n^{0.387})$ colors.

The best known algorithm does not use much fewer than $\tilde{O}(n^{0.387})$ colors as graph coloring is very difficult.

Fact 1

A 2-colorable graph can be 2-colored in polynomial time.

Fact 2

An arbitrary graph with maximum degree Δ can be $(\Delta + 1)$-colored in polynomial time.

Algorithm: $O(\sqrt{n})$ Coloring of 3-Colorable Graphs

1. While there exists a vertex v with $\deg(v) \geq \sqrt{n}$,
   - Pick 3 new colors and color v with one color, and its neighbors with a 2-coloring algorithm.
   - Remove v and $N(v)$ from G.
2. Use the $(\Delta + 1)$-coloring algorithm to color rest of graph with degree $< \sqrt{n}$.

Theorem

This algorithm colors any 3-colorable graph with at most $4\sqrt{n}$ colors.

Proof. Each time the algorithm finds a vertex of degree $\geq \sqrt{n}$, it uses three new colors. This can happen at most $n/\sqrt{n}$ times since we remove $\sqrt{n}$ vertices each iteration. This uses $3\sqrt{n}$ colors.

The final step uses an additional $\sqrt{n}$ colors for a total of $4\sqrt{n}$ colors. ■

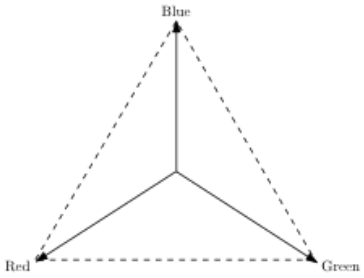
We can do better using SDPs. Consider the following VP where we have v_i for each $i \in V$.

$$\begin{aligned} \min \quad & \lambda \\ \text{subject to} \quad & v_i \cdot v_j \leq \lambda, \quad \forall (i,j) \in E \\ & v_i \cdot v_i = 1, \quad \forall i \in V \\ & v_i \in \mathbb{R}^n, \quad \forall i \in V \end{aligned} \qquad \textbf{(Coloring VP)}$$

Lemma

For any 3-colorable graph, there is a feasible solution to **Coloring VP** with $\lambda \leq -\frac{1}{2}$.

Proof. Consider an equilateral triangle and associate the unit vectors for all the vertices with the three different colors with the three different vertices of the triangle.



The angle between any two vectors of the same color is 0 and the angle between vectors of different color is $2\pi/3$ or $120°$. Then for $(i, j) \in E$, we have

$$v_i \cdot v_j = \|v_i\| \, \|v_j\| \cos(2\pi/3) = -\frac{1}{2}$$

Since we have a feasible solution to the VP with $\lambda = -\frac{1}{2}$, it must be the case for the optimal solution to have $\lambda \leq -\frac{1}{2}$. ■

The proof of the lemma actually gives the following corollary.

Corollary

For any 3-colorable graph, there is a feasible solution to **Coloring VP** such that $v_i \cdot v_j = -\frac{1}{2}$ for all edges $(i, j) \in E$.

Definition: Semicoloring

A coloring of the vertices such that at most $n/4$ edges have endpoints with the same color.

The randomized algorithm will produce a semicoloring. This implies that at least $n/2$ vertices are colored such that any edge between them has endpoints that are different colors.

A semicoloring will be sufficient. Once we semicolor a graph with k colors, then we can color the entire graph with $k \log n$ colors.

Algorithm: $\tilde{O}(\Delta^{\log_3 2}) = \tilde{O}(n^{\log_3 2})$ Randomized Coloring

1. Semicolor the graph with k colors.
2. Semicolor the vertices that were incorrectly colored with k new colors.
3. Repeat Step 2 ($\log n$ iterations).

Algorithm: Semicoloring

1. Solve VP **Coloring VP**.
2. Choose $t = 2 + \log_3 \Delta$ random vectors $r_1, \ldots, r_t$ where $\Delta = \max_{v \in V} \deg(v)$.
3. t random vectors define 2^t different regions v_i can fall (one region for $r_j \cdot v_i \geq 0$ and one region for $r_j \cdot v_i < 0$ for all $j = 1, \ldots, t$).
4. Color the vectors in each region with a distinct color.

Theorem

The coloring algorithm produces a semicoloring of $4\Delta^{\log_3 2}$ colors with probability $\geq \frac{1}{2}$.

Proof. Since 2^t colors were used for $t = 2 + \log_3 \Delta$, the number of colors used is

$$2^{2+\log_3 \Delta} = 4 \cdot 2^{\log_3 \Delta} = 4\Delta^{\log_3 2}$$

We need to show this produces a semicoloring with probability $\geq \frac{1}{2}$. First consider the probability that vertices i and j get the same color for a given edge (i, j). This probability is the probability that both i and j fall in the same region. Recall the probability that a single hyperplane separates i and j is $\theta_{ij}/\pi = \arccos(v_i \cdot v_j)/\pi$. Therefore, the probability that t independently chosen hyperplanes fail to separate i and j or that they both get the same color is

$$\Pr[i \text{ and } j \text{ are same color for edge } (i,j)] = \left(1 - \frac{1}{\pi}\arccos(v_i \cdot v_j)\right)^t \leq \left(1 - \frac{1}{\pi}\arccos(\lambda)\right)^t$$

where the last inequality follows from arccos being a nonincreasing function and the SDP constraint. Then by previous lemma,

$$\left(1 - \frac{1}{\pi}\arccos(\lambda)\right)^t \leq \left(1 - \frac{1}{\pi}\arccos\left(-\frac{1}{2}\right)\right)^t = \left(1 - \frac{1}{\pi} \cdot \frac{2\pi}{3}\right)^t = \left(\frac{1}{3}\right)^{2+\log_3 \Delta} \leq \frac{1}{9\Delta}$$

Let $m = |E|$, then $m \leq n\Delta/2$. Thus, the expected number of edges which have both endpoints colored the same is no more than $m/9\Delta \leq n/18$. Let X be a random variable denoting the number of edges with both endpoints colored the same. By Markov's inequality, the probability that there are more than $n/4$ edges which have both endpoints colored the same is at most

$$\Pr[X \geq n/4] \leq \frac{E[X]}{n/4} \leq \frac{n/18}{n/4} \leq \frac{1}{2}$$

■

If we use n as a bound on Δ, then we obtain an algorithm that produces a semicoloring with $O(n^{\log_3 2})$ colors and thus a coloring with $\tilde{O}(n^{\log_3 2}) = \tilde{O}(n^{0.631})$ colors. This is worse than the $O(\sqrt{n})$ coloring, but we can use ideas from that coloring algorithm to improve this.

Let σ be a parameter such that as long as there is a vertex in the graph of degree at least σ, we pick 3 new colors and do the 2-coloring on the neighbors of that vertex and remove these vertices. When all vertices in the graph have degree $< \sigma$, we use the semicoloring SDP rounding algorithm to semicolor the remaining graph with $O(\sigma^{\log_3 2})$ colors.

Algorithm: $\tilde{O}(n^{0.387})$ Randomized Coloring

1. Let $\sigma = n^{\log_6 3}$.
2. While there exists a vertex v with $\deg(v) \geq \sigma$,
   - Pick 3 new colors and color v with one color and its neighbors with a 2-coloring.
   - Remove v and $N(v)$ from G.
3. Use the $\tilde{O}(n^{\log_3 2}) = \tilde{O}(\sigma^{\log_3 2})$-randomized coloring algorithm to color the rest of the remaining graph.

Theorem

The algorithm semicolors a 3-colorable graph with $O(n^{0.387})$ colors with probability at least $\frac{1}{2}$.

Proof. The first part of the algorithm uses $3n/\sigma$ colors in total. To balance the two parts of the algorithm, set σ such that $n/\sigma = \sigma^{\log_3 2}$, which gives $\sigma = n^{\log_6 3} \approx n^{0.613}$. Thus, both parts use $O(n^{0.387})$ colors. ■

From this theorem, we get an overall algorithm that colors using $\tilde{O}(n^{0.387})$ colors.

Chapter 9

Lagrangian Relaxation

9.1 k-Median

9.2 k-Minimum Spanning Tree

Chapter 10

Hardness of Approximation