CMPUT 605 Approximation Algorithms Individual Study

Keven Qiu
Instructor: Zachary Friggstad
Fall 2025

Contents

Chapter 1

Classic Approximations

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,

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

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

Proof. Let $e_1, \ldots, e_n$ be the order the algorithm covers the e_i's. Consider the time before e_k is covered. The remaining $n - k + 1$ elements can be covered at a price/cost of no more than OPT. We can cover each element at a cost of no more than $\frac{\text{OPT}}{n-k+1}$ on average.

Suppose not, that is we cannot cover the rest of each element at a cost of no more than $\frac{\text{OPT}}{n-k+1}$ on average. Then the cost of covering the rest of the elements is $> (n-k+1) \cdot \frac{\text{OPT}}{n-k+1} = \text{OPT}$ which contradicts that the rest of the elements can be covered by $\leq \text{OPT}$. ■

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 $\leq 2 \cdot \mathrm{OPT}$.

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 $\leq \text{cost(T')}/2 \leq \text{OPT}/2$ 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; $d_{jj'} \leq d_{jj_i} + d_{j_ij'} = 2r^*$.

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.

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

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,

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

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

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

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

■

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 pseudo-polynomial 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, $\text{OPT}(I) < p(|I_u|)$. If Π admits an FPTAS, then it also admits a pseudo-polynomial 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

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

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 pseudo-polynomial 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 pseudo-polynomial 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: 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

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

$(*)$ 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

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

■

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 $\le T$ makespan exists).

Let b be the smallest integer such that $1/b \le \varepsilon$. For $\varepsilon \le 1$, we have $b \ge 2/\varepsilon$ (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

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

and further, $p'_j = mT/b^2$ for some $m \in \{b, b+1, \ldots, b^2\}$ ($m \ge b$ since $p_j \ge T/b$ 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

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

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 $n_b, \ldots, n_{b^2} \ge 0$ 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

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

For all i, $n_i \le n$ 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 $f(\overline{n}_b, \ldots, \overline{n}_{b^2}) \le k$, output YES. Otherwise, output NO. Each machine is assigned at most b jobs since $p'_j \ge T/b$ for all $j \in J_{large}$, and the p'-makespan is $\le T$. Therefore, true makespan is

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

■

Chapter 3

Linear Programming

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.

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

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

Theorem (LP-Duality)

The primal program has finite optimum if and only if its dual has finite optimum. Moreover, if $x^* = (x_1^*, \dots, x_n^*)$ and $y^* = (y_1^*, \dots, y_m^*)$ 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 $\mathbf{NP} \cap \text{co} - NP$.

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

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

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

Proof.

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

where $\text{OPT} \leq$ 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 $\text{OPT} \leq \text{OPT}_f$.

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.

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

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 $\geq \frac{3}{4}\text{OPT}$.

Proof.

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

■

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.

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

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

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

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

1. Let x be an optimal solution to $\mathbf{MC-LP}$.
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.

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

■

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,

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

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,

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

■