CMPUT 605 Approximation Algorithms Individual Study

Keven Qiu
Instructor: Zachary Friggstad
Fall 2025

Contents

1 Classic Approximations **2**

1.1 Vertex Cover 2

1.2 Set Cover 3

1.3 Steiner Tree 4

1.4 Traveling Salesman Problem 5

1.5 Multiway Cut and k-Cuts 7

1.6 k-Center 7

1.7 Scheduling Jobs on Parallel Machines 8

2 Polynomial-Time Approximation Schemes **10**

2.1 Knapsack 10

2.2 Strong **NP**-Hardness and Existence of FPTAS 12

2.3 Bin Packing 12

2.4 Minimum Makespan Scheduling 13

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 polynomialt ime computable lower bound ont he 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, $|M| \leq \mathrm{OPT}$. Our cover picked has cardinality $2\,|M| \leq 2 \cdot \mathrm{OPT}$. ∎

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. 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 $\leq$ 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 $G \le$ 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 $c_G(uv) \le c_{G'}(uv)$. ■

■

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 $\le 2 \cdot \text{OPT}$.

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 $2 - \frac{2}{k}$-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 $2 - \frac{2}{k}$-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 $S = \{a_1, \ldots, a_n\}$ of objects, with specified sizes and profits in $\mathbb{Z}^+$ and a knapsack capacity $B \in \mathbb{Z}^+$, find a subset of objects whose total size 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 profitable object, $P = \max_{a \in S} p(a)$, then nP is the upperbound on the profit of any solution. Let $S_{i,p}$ be the subset of $\{a_1, \ldots, a_i\}$ whose total profit is exactly p and whose total size is minimized. Let $A(i,p)$ be the size of the set $S_{i,p}$. $A(1,p)$ is known for $\{0, \ldots, nP\}$.

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

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 $P = \max_{a \in S} p(a)$, let $K = \frac{\varepsilon P}{n}$.
2. $p'(a_i) = \left\lfloor \frac{p(a_i)}{K} \right\rfloor$.
3. Solve Knapsack with Dynamic Programming on new profits to get S'.

Lemma

Let set S' outputted by the FPTAS satisfies

$$p(S') \geq (1 - \varepsilon) \cdot \text{OPT}$$

Proof. Let O be the optimal set. For any object a, because we round down, $K \cdot p'(a) \leq p(a)$, but not more than K. Therefore,

$$p(O) - Kp'(O) \leq nK$$

for all objects a. The DP step must return a set at least as good as O under the new profits and multiplying both sides by K,

$$p'(S') \geq p'(O) \implies Kp'(S') \geq Kp'(O)$$

Since rounding underestimates, $p(S') \geq Kp'(S')$, so $p(S') \geq Kp'(S') \geq Kp'(O)$. Thus,

$$p(S') \geq Kp'(O) \geq p(O) - nK = \text{OPT } - \varepsilon P \geq (1 - \varepsilon) \cdot \text{OPT}$$

where last inequality follows from OPT $\geq P$. ■

Theorem

The algorithm is a FPTAS for Knapsack.

Proof. By lemma, the solution found is within $(1 - \varepsilon)$ of OPT . Since the running time of the algorithm is $O(n^2 \lfloor P/K \rfloor) = O(n^2 \lfloor n/\varepsilon \rfloor) = O(n^3/\varepsilon)$, which is polynomial in n and $1/\varepsilon$. ■

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**-hrd 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 the it is not strongly **NP**-hard, assuming $\mathbf{P} \neq \mathbf{NP}$, a contradiction. ■

2.3 Bin Packing

Problem: Bin Packing

Given n items with sizes $a_1, \ldots, a_n \in (0, 1]$, find a packing in unit-sized bins that minimizes number of bins used.

2.4 Minimum Makespan Scheduling