Some More Reductions

6) **Set Cover**: We have a universe $\mathcal{U} = \{1, \ldots, m\}$ and a collection of subsets $S_1, \ldots, S_n$ of $\mathcal{U}$ such that $S_1 \cup \ldots \cup S_n = \mathcal{U}$.

The optimization version of this problem: what is the least number of sets to be taken so that their union is $\mathcal{U}$?

The decision version: Is there a subcollection of k sets $S_{i_1}, \ldots, S_{i_k}$ such that $S_{i_1} \cup \ldots \cup S_{i_k} = \mathcal{U}$?

It is easy to see that Vertex Cover $\leq_p$ Set Cover.
Take $\mathcal{U} = \{e_1, \ldots, e_m\}$ (all the edges in the input graph) and $S_i = \{$edges incident to vertex $i\}$ for each vertex i.

Observe that this instance has a set cover of size k $\iff$ the input graph has a vertex cover of size k.

7) **Integer Programming**: We are given an $m \times n$ matrix A and a column vector b with m entries. Is there an integer-valued vector x such that

$$Ax \leq b \; ?$$

It is easy to see that 3SAT $\leq_p$ Integer Programming.
Given a 3CNF formula ϕ, we can write it down as an integer program such that ϕ is satisfiable $\iff$ our integer program has an integral solution.

Exercise. Please show the above reduction.

Introduction to Approximation Algorithms

Vertex Cover: This problem has an easy 2-approximation algorithm.

1. Find a maximal matching M in the input graph G.
2. For each edge e in M: include both endpoints of e in our set C.

Claim. C is a vertex cover. (Please show this.)

We now claim that $|C| \leq 2k$ where k = size of minimum vertex cover in G. This is because $k \geq |M|$ and $|C| = 2|M|$.

Suppose every vertex has a weight associated with it. We want to find a minimum-weight vertex cover. Let us write the LP for fractional vertex cover.

$$\min \sum_{u \in V} w_u \cdot x_u$$

subject to

$$x_u + x_v \geq 1 \quad \text{for each edge } (u,v)$$
$$x_u \geq 0 \quad \text{for every vertex } u.$$

The approximation algorithm:

1. Solve the above LP. Let x^* be the optimal solution.
2. For each vertex u do: if $x^*_u \geq 1/2$ then add u to our vertex cover C; else do not add u to C.

Claim. The set C is a vertex cover.

For every edge (u,v), since $x^*_u + x^*_v \geq 1$, at least one of x^*_u, x^*_v is $\geq 1/2$. Thus the set C is indeed a vertex cover.

Claim. $w(C) \leq 2 \cdot w(OPT)$, where OPT is a min-wt. vertex cover.

Observe that $w(OPT) \geq w(x^*)$ and $w(C) \leq 2 \cdot w(x^*)$. Thus $w(C) \leq 2 \cdot w(OPT)$.

Set Cover - A greedy algorithm

0. Initially all the elements in the universe $U = \{1, \ldots, m\}$ are uncovered.
1. $C = \phi$.
2. while there exist one or more uncovered elements do:
   - find the set S that minimizes the ratio
   $$\frac{\text{cost}(S)}{\text{no. of uncovered elmts in } S}$$
   - add the set S to C.
3. Return C.

Claim. $|C| \leq (\ln m) \cdot \text{opt}$, where opt is the cost ~~size~~ of a minimum cost set cover.

Renumber the elements of U in the order that they got covered in our algorithm, where ties are resolved arbitrarily. Let $e_1, \ldots, e_m$ be this numbering.

Define $\text{price}(e_i) = \dfrac{\text{cost}(S)}{\text{no. of uncovered elmts in } S \text{ at the time } S \text{ got picked}}$

where S is the set that covers e_i for the first time in our algorithm.

$$\text{cost}(C) = \sum_{S \in C} \text{cost}(S) = \sum_{i=1}^{m} \text{price}(e_i).$$

Claim. For every $i \in \{1, \ldots, m\}$, we have $\text{price}(e_i) \leq \frac{OPT}{m-i+1}$.

Proof. Let $\{S_1, \ldots, S_k\}$ be the optimal set cover. So $c(S_1) + \ldots + c(S_k) = opt$. This means that

$$|S_1| \cdot \frac{c(S_1)}{|S_1|} + \ldots + |S_k| \cdot \frac{c(S_k)}{|S_k|} = opt.$$

Since $|S_1| + \ldots + |S_k| \geq m$, there has to be some j such that $|S_j| \cdot \frac{c(S_j)}{|S_j|} \leq \frac{opt}{m}$.

(otherwise the LHS above will be $> (|S_1| + \ldots + |S_k|) \cdot \frac{opt}{m} \geq opt$.)

So in the first step of the algo., when we choose a set S that minimizes $\frac{c(S)}{|S|}$, we have $\text{price}(e_1) \leq \frac{opt}{m}$.

Consider any later iteration in the algorithm where elements $e_1, \ldots, e_{k-1}$ have been covered and we are now picking a set S' that covers $e_k, e_{k+1}, \ldots$
We will show $\text{price}(e_k) \leq \frac{OPT}{m-k+1}$.

It is the same idea as in the first iteration. Delete $e_1, \ldots, e_{k-1}$ from the universe. Let $S'_1, \ldots, S'_r$ be the non-empty sets in the optimal set cover now. Using our old reasoning, for at least one of these sets S'_j we have

$$\frac{c(S'_j)}{|S'_j|} \leq \frac{opt}{m-k+1}$$ where $|S'_j| =$ number of uncovered elmts in S'_j. ∎

Thus the cost of our set cover $C = \sum_{i=1}^{m} price(e_i)$

$$\leq opt. \left(\frac{1}{m} + \frac{1}{m-1} + \ldots + 1\right)$$

$$= opt. H_m \approx (\ln m). opt.$$

This proves the approximation guarantee of the greedy algorithm for set cover.

The TSP (Traveling salesman problem)

Given a complete graph with non-negative edge costs, find a minimum cost cycle visiting every vertex exactly once.

We will now show that the above problem cannot be approximated within any multiplicative factor in polynomial time, assuming $P \neq NP$.

Claim. TSP cannot be approximated within a multiplicative factor of c in polynomial time, for any c, unless $P = NP$.

Proof. Suppose not. We will use such an algorithm to solve the Hamiltonian cycle problem in polynomial time. However the Hamiltonian cycle problem is NP-hard. Thus a polynomial time algorithm to solve the Hamiltonian cycle problem implies $P = NP$.

Assign a weight of 0 to the edges of G (the input to the Hamiltonian cycle problem) and all edges missing in G are also added - with a weight of 1 on each such edge - to obtain a complete graph with non-negative edge weights.

We will feed this graph as an input to the approximation algorithm.

- If G has a Hamiltonian cycle, then there is a cycle of weight or cost 0 in our complete graph, visiting every vertex exactly once. So the approximation algo. has to find a cycle of cost $\leq c.0 = 0$.
- Else any cycle in the complete graph that visits every vertex exactly once has cost ≥ 1. So the approximation algo. returns a cycle of cost ≥ 1.

Thus we can solve the Hamiltonian cycle problem in polynomial time. ▨

In order to obtain such a strong inapproximability result, we had to assign edge costs that violate triangle inequality. If we restrict ourselves to graphs where edge costs satisfy triangle inequality, the resulting problem is called metric TSP.

Metric TSP is also NP-hard. But it is no longer hard to approximate.

A simple 2-approximation algorithm

1. Find a minimum spanning tree T in G.
2. Double every edge of T to obtain an Eulerian graph.
3. Find an Eulerian tour S on this graph.
4. Output the tour that visits vertices of G in the order of their first appearance in S.

Let C be the tour returned by the above algorithm.

Claim. $\text{cost}(C) \leq 2 \cdot \text{opt}$, where opt is the cost of the optimal tour.

Proof. We have $\text{cost}(T) \leq \text{opt}$ since deleting an edge from the optimal tour gives us a spanning tree in G. Since the Euler tour S contains each edge of T twice, $\text{cost}(S) = 2 \cdot \text{cost}(T)$.

Because of triangle inequality, after the short-cutting step (Step 4), $\text{cost}(C) \leq \text{cost}(S)$. Thus $\text{cost}(C) \leq 2 \cdot \text{opt}$. ▨

An improved approximation algorithm

Is there a cheaper Euler tour than that found by doubling an MST? Recall that a graph has an Euler tour if and only if all its vertices have even degrees. Thus we only need to be concerned about the vertices of odd degree in the MST.

A $3/2$-approximation algorithm

1. Find an MST of G, say T.
2. Compute a min-cost perfect matching M on the set of odd degree vertices of T. Add M to T and obtain an Eulerian graph.
3. Find an Euler tour S of this graph.
4. Output the tour that visits vertices of G in order of their first appearance in S. Let C be this tour.

Claim. Let $V' \subseteq V$ such that $|V'|$ is even, and let M be a min-cost perfect matching on V'. Then $\text{cost}(M) \leq \text{opt}/2$.

Proof. Consider an optimal TSP tour of G, say τ. Let τ' be the tour on V' obtained by short-cutting τ. By the triangle inequality, $cost(\tau') \leq cost(\tau)$. Observe that τ' can be regarded as the union of two perfect matchings on V', each consisting of alternate edges of τ'. Thus the cheaper of these matchings has cost $\leq \frac{cost(\tau')}{2} \leq \frac{opt}{2}$. Hence the optimal matching also has cost at most $opt/2$. ▨

Claim. The above algorithm achieves an approximation guarantee of $3/2$ for metric TSP.

Proof. The cost of the Euler tour S can be bounded as follows:

$$cost(S) \leq cost(T) + cost(M)$$
$$\leq opt + opt/2$$
$$= \frac{3}{2} opt.$$

Using the triangle inequality, $cost(C) \leq cost(S)$. Thus $cost(C) \leq \frac{3}{2} \cdot opt$. ▨

Knapsack

The knapsack problem is an NP-hard optimization problem that allows approximability to any required degree.

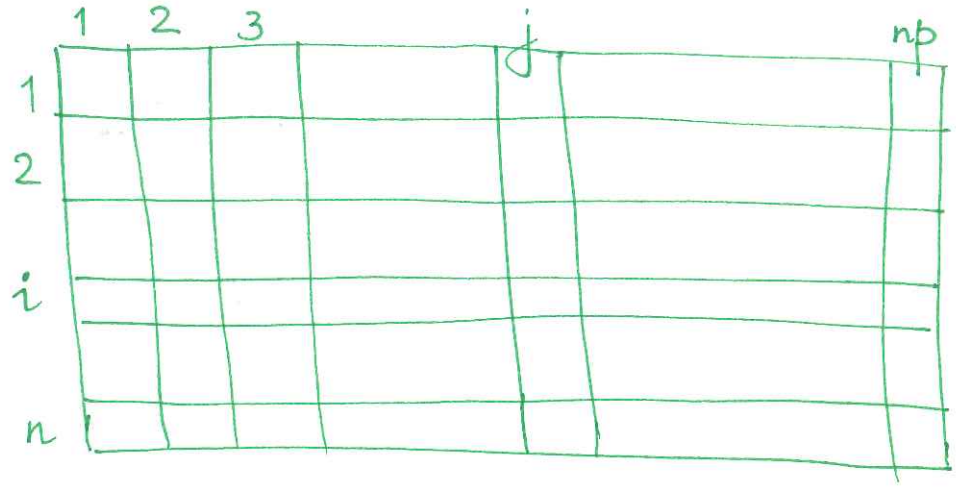
Knapsack: Given a set $S = \{a_1, \ldots, a_n\}$ of objects, where each object has a size and a profit (both these are non-negative integers) along with a "knapsack capacity" $B \in \mathbb{Z}^+$, find a subset of objects whose total size is at most B and total profit is maximized.

We will see that knapsack admits a fully polynomial time approximation scheme (FPTAS). That is, given an input instance I and error parameter ε, such an approximation algorithm returns an answer $\geq (1-\varepsilon) \cdot opt$ and its running time is $poly(n, \frac{1}{\varepsilon})$.

To begin with, let us show a pseudo-polynomial time algorithm for knapsack. That is, the running time of this algorithm will be polynomial in n and in profit values. Note that these values are written in binary, so a running time that is polynomial in these values is not a polynomial time algorithm. However it tells us that if the profit values are $poly(n)$, then the knapsack problem admits a polynomial time algorithm.

The pseudo-polynomial time algorithm runs as follows: let p be the profit of the most profitable object, i.e., $p = \max_{a \in S} profit(a)$.

We will build an $n \times np$ size table A.



For any $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, np\}$, let $S_{i,j}$ denote a subset of $\{a_1, \ldots, a_i\}$ whose total profit is exactly j and whose total size is minimized.

Let $A[i, j]$ be the size of $S_{i,j}$ if such a set exists; else $A[i, j] = \infty$.

The following recurrence helps compute all values $A[i, j]$ in $O(n^2 p)$ time.

$$A[i,j] = \begin{cases} \min\{A[i-1, j],\ size(a_i) + A[i-1, j - profit(a_i)]\} & \\ \qquad\qquad \text{if } profit(a_i) \leq j & \\ A[i-1, j] \qquad \text{otherwise.} & \end{cases}$$

Clearly $A[1, j]$ is known for every $j \in \{0, 1, \ldots, np\}$. Thus we can determine all entries in the above table.

The maximum profit achievable by objects of total size at most B is $\max\{j : A[n, j] \leq B\}$. Thus we have a psendo-polynomial time algorithm for knapsack.

An FPTAS for knapsack

The main idea here is to modify the profits to make them poly$(n, \frac{1}{\varepsilon})$. This will allow us to use the dynamic programming algo. to find the most profitable set as per the modified profits.

The algorithm (I/P: a knapsack instance and $\varepsilon > 0$)

1. For each object a_i, define $profit'(a) = \left\lfloor \frac{profit(a) \cdot n}{\varepsilon p} \right\rfloor$
2. With the modified profits (as given by profit'), find the most profitable set, say X.
3. Return X.

Claim. $profit(X) \geq (1-\varepsilon) \cdot opt$.

Proof. For any object a, we have:

$$\frac{profit(a) \cdot n}{\varepsilon p} - 1 \leq profit'(a) \leq \frac{profit(a) \cdot n}{\varepsilon p}.$$

Thus
$$\begin{aligned} profit(X) &\geq \frac{\varepsilon p}{n} \cdot profit'(X) \\ &\geq \frac{\varepsilon p}{n} \cdot profit'(OPT) \\ &\geq \frac{\varepsilon p}{n} \left(\frac{profit(OPT) \cdot n}{\varepsilon p} - n \right) \\ &= profit(OPT) - \varepsilon p \\ &\geq opt - \varepsilon \cdot opt \quad (\text{since } p \leq opt) \\ &= (1-\varepsilon) \cdot opt. \end{aligned}$$

∎

Since the running time of the above algorithm is poly$(n, \frac{1}{\varepsilon})$, we have an FPTAS for the knapsack problem.