1 Analysis of Christofides’ Algorithm

Theorem 1. Christofides’ algorithm is a $1.5$-approximation algorithm for metric TSP.

Proof. Polynomial running time and correctness: Refer to Lecture 1.

Now we show that the weight of the output solution is at most $1.5$ times the weight of an optimal TSP solution $OPT$.

- The weight of the minimum spanning tree $T$ is at most $OPT$; (see Lecture 1)
- The weight of the minimum cost perfect matching $M$ is at most $OPT/2$.

To prove the second point above, we show that there exists a perfect matching of weight at most $OPT/2$:

a) Start from an optimal TSP tour of the entire graph;
b) “Shortcut” the tour so that it visits only the odd degree vertices of $T$; (the new tour again has weight at most $OPT$ from the properties of “shortcut”);
c) Color the edges of the new tour using red and blue colors alternatively;
d) The set of the red edges (resp. the blue edges) is a perfect matching, and one of the two sets has weight at most $OPT/2$.

2 The Knapsack Problem

Input: capacity $B \in \mathbb{N}$; $n$ items where the $i^{th}$ item has value $v_i \in \mathbb{N}$ and size $s_i \in \mathbb{N}$. (We assume that $s_i \leq B$ for every $i$.)

Output: a subset of items $S$ that maximizes the total value $\sum_{i \in S} v_i$ such that the total size $\sum_{i \in S} s_i$ is at most $B$.

2.1 Dynamic Program

For each $j \in \{1, \ldots, n\}$, maintain a list $A(j)$ consisting of pairs $(t, w)$ such that there is a subset of the first $j$ items with total size $t$ and total value $w$. We say that a pair $(t, w)$ dominates another pair $(t', w')$ if $t \leq t'$ and $w \geq w'$. If a pair is dominated by some other pair, we remove it from the list $A(j)$. Therefore, we can assume that the list $A(j)$ consists of the pairs $(t_1, w_1), \ldots, (t_k, w_k)$ such that $t_1 \leq t_2 \leq \cdots \leq t_k$ and $w_1 \leq w_2 \leq \cdots \leq w_k$. Let $V = \sum_i v_i$ be the maximum possible value for the knapsack. Every list $A(j)$ contains at most $\min(B + 1, V + 1)$ pairs. The lists $\{A(j)\}_j$ are computed in Algorithm 1.

Correctness: Algorithm 1 computes the optimal value of the knapsack problem. (Key: For every $j \in \{1, \ldots, n\}$, the list $A(j)$ contains all non-dominated pairs corresponding to feasible subsets of the first $j$ items, and the optimal solution corresponds to a non-dominated pair when $j = n$.)

Running time: $O(n \cdot \min(B, V))$, which is exponential in the size of the input.
Algorithm 1 Dynamic program for the knapsack problem

1: \[ A(1) \leftarrow \{(0, 0), (s_1, v_1)\} \]
2: \textbf{for} \( j \leftarrow 2 \) to \( n \) \textbf{do}
3: \[ A(j) \leftarrow A(j - 1) \]
4: \textbf{for} \( (t, w) \in A(j - 1) \) \textbf{do}
5: \quad \text{if} \( t + s_j \leq B \) then
6: \quad \quad \text{Add} \( (t + s_j, w + v_j) \) to \( A(j) \)
7: \quad \text{Remove dominated pairs from} \( A(j) \)
8: \textbf{return} maximum value \( w \) such that \( (t, w) \in A(n) \)

2.2 Approximation Scheme

\textbf{Definition 1.} A polynomial-time approximation scheme (PTAS) is a family of polynomial-time algorithms \( A_\varepsilon \), where there is an algorithm for each \( \varepsilon > 0 \) such that \( A_\varepsilon \) is a \((1 + \varepsilon)\)-approximation algorithm. A fully polynomial-time approximation scheme (FPTAS) is a PTAS such that the running time of \( A_\varepsilon \) has a polynomial dependence in \( 1/\varepsilon \).

To obtain a FPTAS for knapsack, we round the value \( v_i \) of each item \( i \) to some multiple of \( \mu \) (where \( \mu \) is a parameter to be defined).

Algorithm 2 Approximation scheme for the knapsack problem

1: \( M \leftarrow \max_i v_i \)
2: \( \mu \leftarrow \varepsilon M / n \)
3: \( v'_i \leftarrow \lfloor v_i / \mu \rfloor \) for each \( i \)
4: Apply Algorithm 1 on the knapsack instance with values \( \{v'_i\}_i \)

\textbf{Theorem 2.} Algorithm 2 is a fully polynomial-time approximation scheme for the knapsack problem.

\textbf{Proof.} First, we show that the algorithm returns a solution with value at least \((1 - \varepsilon)\) times the value of an optimal solution \( \text{OPT} \). Let \( M \) be the maximum value of an item. Clearly, \( \text{OPT} \geq M \) since one possible solution is to select the most valuable item alone in the knapsack. From the definition of the rounding, we have \( \mu v'_i \leq v_i \leq \mu(v'_i + 1) \), thus \( \mu v'_i \geq v_i - \mu \). Let \( S \) be the set of items returned by Algorithm 2, and let \( O \) be the set of items in the optimal solution. We have:

\[
\sum_{i \in S} v_i \geq \mu \sum_{i \in S} v'_i \\
\geq \mu \sum_{i \in O} v'_i \\
\geq \sum_{i \in O} (v_i - \mu) \\
\geq \left( \sum_{i \in O} v_i \right) - n \cdot \mu \\
= \left( \sum_{i \in O} v_i \right) - \varepsilon \cdot M \\
\geq \text{OPT} - \varepsilon \cdot \text{OPT}.
\]

Next, we show that the running time of the algorithm is polynomial in \( n \) and in \( 1/\varepsilon \). When we apply Algorithm 1 with values \( \{v'_i\}_i \), since each \( v'_i \) is at most \( M/\mu = n/\varepsilon \), the total value \( V' := \sum_i v'_i \) is at most \( n^2/\varepsilon \). Thus the running time of applying Algorithm 1 on this instance is \( O(n \cdot \min(B, V')) = O(n^2/\varepsilon) \).
3 The Set Cover Problem

Input: a ground set of elements $E = \{e_1, \ldots, e_n\}$, subsets of those elements $S_1, \ldots, S_m$ where each $S_j \subseteq E$, and a nonnegative weight $w_j \geq 0$ for each subset $S_j$.

Output: minimum weight collection of subsets that covers all elements of $E$.

3.1 Greedy Algorithm

Algorithm 3 Greedy algorithm for the set cover problem

1: $I \leftarrow \emptyset$
2: $\hat{S}_j \leftarrow S_j$ for all $j$
3: while $I$ is not a set cover do
4: Let $\ell$ be the set such that $w_\ell / |\hat{S}_\ell|$ is minimized
5: $I \leftarrow I \cup \{\ell\}$
6: $\hat{S}_j \leftarrow \hat{S}_j \setminus S_\ell$ for all $j$

return $I$

Let $H_k$ denote the $k^{th}$ harmonic number, i.e., $H_k = 1 + (1/2) + (1/3) + \cdots + (1/k)$. We know that $H_k \approx \ln k$.

Theorem 3. Algorithm 3 is an $H_n$-approximation algorithm for the set cover problem.

Proof. Let $n_k$ be the number of elements that remain uncovered at the beginning of the $k^{th}$ iteration. If the algorithm takes $\ell$ iterations, then $n_1 = n$ and $n_{\ell+1} = 0$. We claim that for each $k \in \{1, \ldots, \ell\}$, the set $j$ chosen in the $k^{th}$ iteration is such that

$$w_j \leq \frac{n_k - n_{k+1}}{n_k} \cdot \text{OPT}. \quad (1)$$

From this claim, we can show the theorem statement. Let $I$ be the final set returned by the algorithm. We have

$$\sum_{j \in I} w_j \leq \sum_{k=1}^{\ell} \frac{n_k - n_{k+1}}{n_k} \cdot \text{OPT}$$

$$\leq \text{OPT} \cdot \sum_{k=1}^{\ell} \left( \frac{1}{n_k} + \frac{1}{n_k-1} + \cdots + \frac{1}{n_{k+1}+1} \right)$$

$$= \text{OPT} \cdot \frac{1}{\sqrt{n}} \left( \sum_{i=1}^{n} \frac{1}{i} \right)$$

$$= H_n \cdot \text{OPT}.$$ 

Now we only need to show the claim in Eq. (1). Consider the $k^{th}$ iteration of the while loop. Let $\{\hat{S}_j\}_j$ be the sets at the beginning of this iteration. Let $O$ be an optimal solution of the set cover instance with weight OPT. In particular, $O$ covers all of the $n_k$ remaining elements. Thus there exists some set $j' \in O$ such that $w_{j'} / |\hat{S}_{j'}| \leq \frac{\text{OPT}}{n_k}$. Let $j$ be the set chosen in this iteration. From the definition, we have $w_j / |\hat{S}_j| \leq \frac{w_{j'}}{|\hat{S}_{j'}|} \leq \frac{\text{OPT}}{n_k}$. Therefore, Eq. (1) follows by observing that $|\hat{S}_j| = n_k - n_{k+1}$. \qed

On the other hand, we can show that there exists some constant $c > 0$ such that if there exists a $c \ln n$-approximation algorithm for the unweighted set cover problem, then $P = NP$. (We don’t prove this statement.)