Problem 1: Bitonic Euclidean TSP

Because no two points have the same x-coordinate, the first step is to sort the n points according to x-coordinate. That is, the leftmost point is named as 1, and the rightmost point is named as n. Let $d(i, j)$ be the distance of path between point i and j, $i, j \in \{1, 2, \ldots, n\}$.

Bitonic tours start at the leftmost point, go strictly rightward to the rightmost point, and then go strictly leftward back to the starting point. And the optimal bitonic tour has no self-crossings, because any two edges that cross can be replaced by an uncrossed pair of edges with shorter total length.

The subproblems:

According to these two properties, we define subproblems as below:

For each subset $S \subset \{1, 2, \ldots, n\}$ such that $1 \in S$ and for each $i, j \in S-\{1\}$, find the minimum total cost of two path where both two path starts at 1 and ends at i and j respectively. The two path cannot visit the same point except for point 1 and point n.

To solve the subproblems, a recursive solution will be introduced. Let $f(i, j)$ be the minimum total cost of two path, then $f(1,1) = 0$ and $f(n, n)$ is final optimal solution which means the minimum total cost of the optimal bitonic tour. Because $f(a, b) = f(b, a)$, we can only consider the situation $1 \leq i \leq j \leq n$. We can define $f(i, j)$ recursively as follows.

Case 1 if $i = j$ or $i = j - 1$, then the optimal path must include a path which starts from some point k with $1 \leq k \leq j$ and ends in point j. Therefore, the optimal solution depends on the optimal choice of point *k*.

$$f(i,j) = \min_{1\le k\le j}\{f(i,k) + d(k,j)\}$$

Case 2 if $i < j - 1$, then the path ending in point j must pass through point $(j - 1)$, and the other path cannot visit point $(j - 1)$.

$$f(i,j) = f(i,j-1) + d(j-1,j)$$

Computing complexity:

The size of subproblems of Case 1 is $O(n)$ with each takes $O(n)$ time. The size of subproblems of Case 2 is $O(n^2)$ with each takes $O(1)$ time. Therefore, total running time is $O(n^2)$.

Backtracking:

In order to recover the optimal bitonic tour, we need to record every $d(k,j)$ in Case 1 and every $d(j\text{-}1,j)$ in Case 2 into a table. Then we can use this table to recover the optimal bitonic tour.

Problem 2: Viterbi Algorithm.

(a)

The subproblems:

Assuming that there are n vertices in G, u, v are the index of vertices, that is, $v \in \{0, 1,2, \ldots, n\text{-}1\}$. Let i indexes the sounds of s, that is $1 \le i \le k$, $\sigma_k(u, v)$ is the labeled edge that begins at u and ends at v. If the required path exists, the optimal solution returns a path that begins at v_0, go through every sound of s, finally ends at v of $\sigma_k(u, v)$. Then, the subproblems can be define as follow:

For each subset $S \subset \{1, 2, \ldots, k\}$ and $V \subset \{0, 1, 2, \ldots, n\text{-}1\}$ such that $j \in S$, $v \in V$ and for each $i \in S\text{-}\{j\}$, $u \in V - \{v\}$, find the path that begins at v_0, go through the edges labeled by sounds in $s - \{\sigma_i\}$, and ends at u of $\sigma_i(u, v)$.

To solve the subproblems, a recursive solution will be introduced. Define $F(i, v) = 1$ if there exists the required path that begins at v_0, go through the sounds before ith in the sequence s, and ends at v of $\sigma_i(u, v)$, and $F(i, v) = 0$ otherwise. The algorithm will return NO-SUCH-PATH if $F(k, v) = 0$, and return $F(i, v) = 1$ otherwise. So the recurrence becomes:

$$F(i, v) = \max\{F(i-1, u) | \sigma_i(u, v) \in s, (u, v) \in E\}$$

Computing complexity:

Because $1 \le i \le k$, $v \in \{0, 1, 2, \ldots, n\text{-}1\}$, the size of subproblems is $O(k \cdot n)$. Each subproblem requires $O(n)$ to compare and maximize the solution.

Therefore, the total time is $O(k \cdot n^2)$.

Backtracking:

In order to backtracking, we need to record the every (u, v) of $\sigma_i(u, v)$ which make the $F(i, v) = 1$. Then, we can recover the path by the records.

(b)

Adding to the probability $p(u, v)$ to the edge (u, v), the subproblem becomes: For each subset $S \subset \{1, 2, \ldots, k\}$ and $V \subset \{0, 1, 2, \ldots, n\text{-}1\}$ such that $j \in S$, $v \in V$ and for each $i \in S\text{-}\{j\}$, $u \in V - \{v\}$, find the maximum probability of the path that begins at v_0, go through the edges labeled by sounds in $s - \{\sigma_i\}$, and ends at u of $\sigma_i(u, v)$. And the value of $F(i, v)$ is not just 0 and 1, it will be the maximum probability of the required path. So the recursive solution becomes:

$$F(i, v) = F(i-1, u) \times \max\{p(u, v) | \sigma_i(u, v) \in s, (u, v) \in E\}$$

The computing complexity and the backtracking method is the same as **(a)**.

Problem 3:

To argue that some bag of T contains C, we prove it by induction.

Define $\boldsymbol{X} = \{X_1, \ldots, X_j, ..., X_n\}$ is a family of subsets of V, and the bags of T are the subsets X_j.

1. This theorem is true for cliques with size 2 by the property that for every edge (v, w) in the graph G, there is a subset X_i that contains both v and w.

2. Suppose this theorem is true for cliques with size k, and now given a clique C_{k+1} with size k +1 such that $C_k = C_{k+1} - \{v\}$. By the hypothesis, there is $\boldsymbol{X}_k \subset \boldsymbol{X}$ such that $\boldsymbol{X}_k$ contains C_k.

(1) If one bag of $\boldsymbol{X}_k$ contains v, the theorem is true because $\boldsymbol{X}_k$ contains C_{k+1}.

(2) If no bag of $\boldsymbol{X}_k$ contains v, then $v \in X_i$, where $X_i \in \boldsymbol{X} - \boldsymbol{X}_k$. Because $\boldsymbol{X}_k$ contains C_k, $\boldsymbol{X} - \boldsymbol{X}_k$ is a forest.

i. If $X_i, X_j \in \boldsymbol{X} - \boldsymbol{X}_k$ are in disconnected part and $v \in X_i \cap X_j$, then all bags X_m of tree in the path between X_i and X_j contain v as well, which contradict the presupposition (2).

ii. If a unique part $\boldsymbol{Y} \subset \boldsymbol{X} - \boldsymbol{X}_k$ contains v, then $\boldsymbol{Y}$ should not contain each vertex $u \in \boldsymbol{X}_k$ because if that happen $\boldsymbol{Y}$ will contain Ck+1 and then the theorem is true. Therefore, if we cut T from the unique edge between $\boldsymbol{Y}$ and $\boldsymbol{X}_k$, then there is the case that $\boldsymbol{Y}$ contain v but no C_k, and $\boldsymbol{X}_k$ contain C_k but no v. However, this case contradict the property of tree decomposition that each edge $\{u, v\}$ must be contained in some bags in T.

As mentioned above, we have proved that some bag of T contains C.

Problem 4:

4(a)

For the tree decomposition T of G, there are three required properties:

1. the union of all set X_i equals V which means each graph vertex is associated with at least one tree node.
2. for every edge (u, v) in the graph, there is a subset X_i that contains both v and w.
3. if X_i and X_j both contain a vertex v, then all nodes X_k of tree in the path between X_i and X_j contain v as well.

In this case, if one bag is a subset of another adjacent bag, we can totally merge this bag into the adjacent bag. After merging the two bags, the 1st property still works. Because the two bags are adjacent, so there is no other node between the two bags such that the 3rd property also works. Moreover, because one bag is the subset of another adjacent bag, the size of this bag must be smaller than the size of the another one. Therefore, T still has treewidth at most k.

4(b)

Given the property that no bag is a subset of an adjacent bag and the 1st property mentioned in 4(a) that each graph vertex in G must be associated with at least one tree node in T, we know that each tree node includes at least one unique vertex that adjacent tree nodes do not contain. If each two tree nodes contains the same unique vertex and they are not adjacent, or there is a subset of tree node which is not adjacent to this tree node, the tree nodes on the path between the two tree nodes must contain the subset because of the 3rd property mentioned in 4(a). Therefore, every tree node cannot have a subset in *T* and every tree node will have its unique vertex, so the size of tree nodes with unique vertex is O(*n*).

4(c)

To get a "smooth" tree, we can take two operations to *T*:

1. Merge all bags with size $t < k$+1 to the bags with size k+1.

2. When all bags' size become k+1, we check every pair of adjacent bags. If the pair of adjacent bags have k vertices in common, we will do nothing. If the pair of adjacent bags X_i, X_j have $m < k$ vertices in common, we insert one bag X_k into the pair of adjacent bags such that X_i connect to X_j through X_k. And X_k contains three parts of vertices. The first part is the m common vertices between X_i, X_j. The second part is m-k vertices from X_i which X_j does not contain. The third part is m-k vertices from X_j which X_i does not contain. Then, the tree *T* will become a "smooth" tree.

Problem 5:

Given a tree decomposition T for G of width k, X_i is a tree node in T. According to the root X_r randomly selected from tree nodes, the T can be rearranged to a bottom-up tree. Because of "smooth", each tree node has its unique vertex which adjacent tree nodes do not contain. Subproblems can be defined as follow.

For each X_i and each vertex cover set $B \subset X_i$, $w(X_i;\, B)$ is the minimum weight of a vertex cover set I in G_i such that $I \cap X_i = B$, where G_i is a subset of G which contains all vertices of X_i and its children. If $B = \phi$, $w(X_i;\, B) = +\infty$.There are two cases.

(1) If X_i has no child, $w(X_i;\, B) = w(B)$, which means $G_i = X_i$, $I = B$.

(2) If X_i has children of size d_i,

$$w(X_i;\ B) = \sum_{X_j} \left[min\left(w(X_j; X_j \cap B), w(X_j; (X_j \cap B) \cup \{u\})\right) - w(B) \right] \\ + w(B)$$

where $j \in \{1, 2, \ldots, d_i\}$ and vertex u is the unique vertex in $\underline{X_j}$ but not in X_i.

For each node X_i, there are at most 2^{k+1} subsets B to consider. Because the size of all tree nodes is k, the time is $O(d_i \cdot k)$ for each subproblem. Because summing up all children up to root must be $O(n)$, the total time is $O(2^k \cdot k \cdot n)$

Actually, the independent set and the vertex cover of G are mutually complementary such that the minimum weighted vertex cover of G is the residue set of G after deleting the maximum weighted independent set.