Math 375 Week 5 Lecture

Sections 3.3 and 3.4

Section 3.3: The Traveling Salesman Problem

Branch and Bound algorithm: Let $C$ be a Hamilton circuit. Then there must be edges of the form $v_1v_{a1}$, $v_2v_{a2}$, etc. The cost of this is $c_{1a1} + c_{2a2} + \cdots$. So at least one number per row is used. Subtracting minimum row number from each row, any Hamilton circuit corresponding to the original matrix is $\Delta +$ the corresponding Hamilton circuit corresponding to the new matrix. In the new matrix, if $C$ is a Hamilton circuit, there must be edges of the form $v_{b1}v_1$, $v_{b2}v_2$, etc. The cost of this is $c_{b1} + c_{b2} + \cdots$. So at least one number per column in the new matrix. So now any Hamilton circuit corresponding to the original matrix is $\Delta + \Delta'$ the corresponding Hamilton circuit corresponding to the new matrix. We will always represent a weight matrix as $\Delta +$ a matrix in which each row and each column contains a 1.

Now look at any entry $c_{ij}$ containing a zero. Have a choice of using that edge or not. If you don’t use it, replace that entry by $\infty$ and adjust $\Delta$. But if you do use it, set $c_{ji} = \infty$ (can’t use edge) and delete row $i$ and column $j$ since nothing else will be chosen from it and adjust $\Delta$. To decide which move to make, go with the smaller $\Delta$.

At every stage have made a decision about whether or not to use an edge, so the process has to terminate.

Now suppose $c_{ij} \leq c_{ik} + c_{kj}$ and $c_{ij} = c_{ji}$ are always true. Then it is possible to construct a Hamilton Circuit with weight less than twice the minimal weight circuit. Method: In principle there is a least cost Hamilton circuit $C$. Remove the costliest edge to create a Hamilton path $P$. Write

$$P = p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_n.$$ 

Then all these weights are $\leq c_{p,np_1}$. We will construct a Hamilton circuit $H$ with $W(H) \leq 2W(C)$. Let $C_0 = p_1 \rightarrow p_1$ and $P_0 = \emptyset$ and $S_0 = C$. Then $W(C_0) \leq 2W(P_0)$. Let $C_1 = p_1 \rightarrow p_1 \rightarrow p_1$, where $c_{p_1p_1}$ is minimal with $p_1 \neq p_1$. Then $W(C_1) = c_{p_1p_1} + c_{p_1p_1} \leq c_{p_1p_2} + c_{p_2p_1} = 2c_{p_1p_2} \leq 2c_{p_1p_1}$. Now remove $p_1$ from $S_0$ to create $S_1$ and set $P_1 = p_1p_n$. In general, assume a cycle $C_k$ has been built with $w(C_k) \leq 2W(P_k)$, where $P_k$ is a subset of $k$ edges of $C$. Let $yz$ be least weight edge connecting vertex in $C_k$ to vertex
not in $C_k$. Let $z'z''$ be first edge in $S_k$ exiting $C_k$ along a path to $z$, and let $y'y$ be edge in $C_k$. To create larger cycle $C_{k+1}$, remove $y'y$ and insert $y'z$ and $zy$. By triangle inequality, 

$$c_{y'z} \leq c_{y'y} + c_{yz}.$$ 

By choice of $yz$, 

$$c_{yz} \leq c_{z'z''}.$$ 

Therefore 

$$W(C_{k+1}) = W(C_k) + c_{y'z} + c_{zy} - c_{y'y} = W(C_k) + (c_{y'z} - c_{y'y}) + c_{zy} \leq W(C_k) + 2c_{zy} \leq W(C_k) + 2c_{z'z''}.$$ 

Remove $z'z''$ from $S_k$ to obtain $S_{k+1}$. Add $z'z''$ to $P_k$ to obtain $P_{k+1}$. We now have $W(C_{k+1}) \leq 2W(P_{k+1})$. When done we have $W(C_n) \leq 2W(P_n) = 2W(C)$.

**Section 3.4: Tree Analysis of Sorting Algorithms**

Bubble Sort: Swap elements out of place from end of list to beginning. This places smallest element first. Takes $n - 1$ comparisons. Now sort remaining elements. $O(n^2)$ comparisons.

Merge Sort: Split list in two, sort the two halves separately, then merge. Obtain roughly $f_n = 2^n + 2f_{n-1}$ comparisons. Write this as a tree with $2^n$ at level 0 and $f_n, f_{n}$ at level 1. Keep on going. Get $n2^n$. If $n$ elements then $O(n \log_2 n)$ operations.

Heap Sort: A heap is elements in a tree structure with the parent always larger than the children. Given list $x_1, \ldots, x_n$, load arbitrarily into tree structure. One element trees are already a heap. Otherwise, make each of the two subtrees a heap, then bubble the top element into place. If the height of the tree is $n$, then elements are bubbled a distance no greater than $n$, so with $2^n$ elements get at most $n2^n$ operations. If $n$ elements then $O(n \log_2 n)$ operations to create a heap. To sort the elements, remove the top element, then bubble the next element up. Takes $O(n \log_2 n)$ operations to remove all elements of heap. In the process, we are sorting the original list.