Notes on Branch and Bound

March 5, 2004

These notes follow the discussion of branch and bound algorithms in Computer Algorithms by E. Horowitz, S. Sahni and S. Rajasekaran.

These notes describe a mathematical model of the process of choosing the next node to expand. This model also includes a function for deciding when to abandon a search path before reaching the end of the path. Abandoning searches early attempts to minimize computational effort to find the minimal solution.

The basis of branch and bound algorithms is a ranking function. The ranking function assigns a value to each node in the graph. At each step, a branch and bound algorithm uses the ranking function to decide which node to expand next. In contrast, the usual DFS and BFS exploration algorithms perform a blind search of the graph.

Ideally, the ranking function, $\hat{c}(x)$, ranks nodes based on the cost of a minimal solution reachable from node x. The problem with this ranking function is that the minimal solution must be known ahead of time.

Instead, the ranking function uses an estimate, $\hat{g}(x)$, of the cost of a minimal solution reachable from node x. Using $\hat{g}(x)$ to rank nodes may require exploring uneccesary nodes in the graph—particularly if $\hat{g}(x)$ is not a good estimate. We saw the results of an imprecise estimate in the job assignment example covered in the 26th lecture. In this example, the entire subtree for the assignment $a : 2$ and $b : 3$ before the the minimal solution is found in a different subtree. In this example, searching the $a : 2, b : 3$ subtree was fairly inexpensive. In other cases, this search may be costly. The final element of the ranking function measures the cost of reaching a node from the root. As the searches gets farther from the root node, the node falls in the ranking.

The function $h(x)$ measures the cost of reaching node x from the root node. After including a function of $h(x)$, the ranking function $\hat{c}(x)$ is

$$\hat{c}(x) = f(h(x)) + \hat{g}(x)$$

in which f determines how much significant to give the cost of reaching node x. If $f(h(x)) \equiv 0$ then the algorithm makes long, deep searches of the graph. If $f(h(x)) > 0$ then the algorithm considers nodes close to the root before making long potentially fruitless forays into the graph.

A node is *live* if its subgraph might contain a minmal solution. Live nodes are potential candidates for exploration. A node is *dead* if its subgraph can not contain a minimal solution. If the estimated cost of a minimal solution reachable from x is less than the actual minimal solution reachable from x then a node can be killed when the estimated cost is greater than the least upper bound on a solution. In symbols, if $c(x) >= \hat{c}(x) >= upper$ then an optimal solution is not reachable from x (assuming *upper* is the least upper bound on a solution).

The job scheduling example in lecture 26 used a least upper bound to kill nodes before searching their subtrees.

It should come as no surprise that tuning $\hat{c}$ by adjusting f and $\hat{g}$ requires emperical studies of representative data sets.

1 Traveling Salesperson Problem

Two branch and bound solutions to the TSP are described in this section. Each solution estimates the cost of a minimal tour using a *reduced cost matrix*. In the first solution, the nodes in a path through the search tree represent cities on a tour. In the second solution, nodes in the search tree represent the inclusion or exclusion of an edge between two cities in the optimal tour.

Suppose the distances between n cities are given in an nxn matrix. A row (or column) in this matrix is reduced if at least one entry in the row is 0 while the others are all positive. A matrix is reduced if all of its rows are reduced. An example of constructing a reduced cost matrix is given in lecture 28.

For a non-leaf node S with parent R, the cost of a tour including a trip from R to S can be estimated by a reduced cost matrix A. Matrix A is constructed by:

1. set all entries of row R and column S to ∞ to avoid making a second visit to R or S in an optimal tour.

2. set $A(S, 1)$ to ∞ to prevent making a premature trip from S to 1.

3. For each row or column that does not contain all ∞, subtract each entry in each row by the smallest value in each row. Subtract each entry in each column by the smallest value in each column.

The estimated length of the shortest tour including a trip from R to S is given by

$$\hat{c}(S) = f(h(R) + 1) + \hat{c}(R) + A(R, S) + r$$

where r is the total value subtracted in step 3. The optimal values of f and h are determined empirically.

Lecture 28 includes an example of solving the TSP using the above cost estimate and tree construction.

In this solution, trees were built by deciding which node to visit next. For example, the nodes in the *ith* level of the graph are the options for the *ith* city in a tour. Another solution can be built by deciding whether or not to include a trip between a particular pair of cities in each step. In this solution, the root node might represent the decision to include or non-include a trip between cities 6 and 8. The left child represents the inclusion this trip and the right child represents the exclusion of this trip. The same distance cost estimate function $\hat{c}$ is used in this solution.

The efficient implementation this algorithm requires a policy for choosing which trip between cities to consider next in the search. A good policy is to choose cities R, S such that a trip between R and S has a high probability of being included in the optimal tour. One way of choosing R and S is to choose cities which result in the highest approximate cost for *not* including a trip between R and S in the tour.

An example of this algorithm is given in lecture 29.

2 Homework

In this homework, you will use both branch and bound solutions for the TSP to solve the following instance of the TSP. Entry $D[i, j]$ in matrix D below contains the distance between city i and j.

$$\begin{bmatrix} 0 & 4 & 9 & 2 & 5 \\ 2 & 0 & 7 & 5 & 1 \\ 8 & 3 & 0 & 6 & 4 \\ 1 & 4 & 9 & 0 & 3 \\ 9 & 2 & 7 & 6 & 0 \end{bmatrix}$$

Part (A). Using $\hat{c}$ given above for TSP, construct a tree showing which nodes are explored in the computation of the shortest tour. In this part, each node represents the choice to travel from the parent through the child (as was done in lecture 27). Show the reduced cost matrix associated with each node. Give the estamited length of a solution for each node. Indicate which nodes are killed before they are expanded.

Part (B). Repeat Part A, but use the binary tree solution discussed in lecture 28. Describe how you decided which pair of cities to include at each step.