Travelling Salesman Problem using Branch and Bound Approach

Chaitanya Pothineni

December 13, 2013

Abstract

To find the shortest path for a tour using Branch and Bound for finding the optimal solutions.A branch-and-bound algorithm consists of a systematic enumeration of all candidate solutions, where large subsets of fruitless candidates are discarded ,by using upper and lower estimated bounds of the quantity being optimized.

1 Introduction

The Traveling Salesman Problem, TSP for short, deals with creating the ideal path that a salesman would take while traveling between cities. The solution to any given TSP would be the cheapest way to visit a finite number of cities,

visiting each city only once, and then returning to the starting point. We also must assume that if there are two cities, city A and city B for example, it costs the same amount of money to travel from A to B as it does from B to A.

For the most part, the solving of a TSP is no longer executed for the intention its name indicates. Instead, it is a foundation for studying general methods that are applied to a wide range of optimization problems.

1.1 Applictions

The TSP naturally arises as a subproblem in many transportation and logistics applications.

Scheduling of a machine to drill holes in a circuit board or other object. In this case the holes to be drilled are the cities, and the cost of travel is the time it takes to move the drill head from one hole to the next.

2 History

The problem was first formulated in 1930 and is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods. Even though the problem is computationally difficult, a large number of heuristics and exact methods are known, so that some instances with tens of thousands of cities can be solved.

The Traveling Salesman Problem (TSP) is a problem whose solution has eluded many mathematicians for years. Currently there is no solution to the TSP that has satisfied mathematicians. Historically, mathematics related to the TSP was developed in the 1800's by Sir William Rowan Hamilton and Thomas Penyngton Kirkman, Irish and British mathematicians,respectively. Hamilton was the creator of the Icosian Game in 1857. It was a pegboard with twenty holes that required each vertex to be visited only once, no edge to be visited more than once, and the ending point being the same as the starting point.This kind of path was eventually referred to as a Hamiltonian circuit.However, the general form of the TSP was first studied by Karl Menger in Vienna and Harvard in the late 1920's or early 1930's.

TSP?s were first studied in the 1930?s by mathematician and economist Karl Menger in Vienna and Harvard. It was later investigated by Hassler Whitney and Merrill Flood at Princeton.

In 1994, Applegate, Bixby , Chvatal, and Cook solved TSP containing 7,397 cities. Later in 1998, they solved it using 13,509 cities in United States. In 2001, Applegate, Bixby, Chvtal, and Cook found the optimal tour of 15,112 cities in Germany. Later in 2004, TSP of visiting all 24,978 cities in Sweden was solved; a tour of length of approximately 72,500 kilometers was found and it was proven that no shorter tour exists. This is currently the largest solved TSP. [1]

3 Optimal Solution for TSP using Branch and Bound

A branch-and-bound algorithm consists of a systematic enumeration of all candidate solutions, where large subsets of fruitless candidate s are discarded ,by using upper and lower estimated bounds of the quantity being optimized.

The Branch and Bound strategy divides a problem to be solved into a number of sub-problems. It is a system for solving a sequence of subproblems each of which may have multiple possible solutions and where the solution chosen for one sub-problem may affect the possible solutions of later sub-problems.

Suppose it is required to minimize an objective function. Suppose that we have a method for getting a lower bound on the cost of any solution among those in the set of solutions represented by some subset. If the best solution found so far costs less than the lower bound for this subset, we need not explore this subset at all.

Let S be some subset of solutions.
L(S)=a lower bound on the cost of *any solution belonging to S*
Let C=cost of the best solution found so far
If $C \leq L(S)$,there is no need to explore S because it does not contain any better solution.
If $C > L(S)$,then we need to explore S because it may contain a better solution. [2]

4 Branch and Bound approach Algorithm

function CHECKBOUNDS(st,des,cost[n][n]) [3] ▷ Cal. the bounds
Global variable: $cost[N][N]$ - the cost assignment.
$pencost[0] = t$
for $i \leftarrow 0, n-1$ **do**
for $j \leftarrow 0, n-1$ **do**
$reduced[i][j] = cost[i][j]$
end for
end for
for $j \leftarrow 0, n-1$ **do**
$reduced[st][j] = \infty$
end for
for $i \leftarrow 0, n-1$ **do**
$reduced[i][des] = \infty$
end for
$reduced[des][st] = \infty$
$RowReduction(reduced)$
$ColumnReduction(reduced)$
$pencost[des] = pencost[st] + row + col + cost[st][des]$
return $pencost[des]$

```
end function
function RowMin(cost[n][n],i)                    ▷ Cal. min in the row
    min = cost[i][0]
    for j ← 0, n − 1 do
        if cost[i][j] < min then
            min = cost[i][j]
        end if
    end for
    return min
end function
function ColMin(cost[n][n],i)                    ▷ Cal. min in the col
    min = cost[0][j]
    for i ← 0, n − 1 do
        if cost[i][j] < min then
            min = cost[i][j]
        end if
    end for
    return min
end function
function Rowreduction(cost[n][n])                ▷ makes row reduction
    row = 0
    for i ← 0, n − 1 do
        rmin = rowmin(cost, i)
        if rmin ≠ ∞ then
            row = row + rmin
        end if
        for j ← 0, n − 1 do
            if cost[i][j] ≠ ∞ then
                cost[i][j] = cost[i][j] − rmin
            end if
        end for
    end for
end function
function Columnreduction(cost[n][n])             ▷ makes column reduction
    col = 0
    for j ← 0, n − 1 do
        cmin = columnmin(cost, j)
        if cmin ≠ ∞ then
            col = col + cmin
        end if
        for i ← 0, n − 1 do
            if cost[i][j] ≠ ∞ then
                cost[i][j] = cost[i][j] − cmin
            end if
        end for
```

```
    end for
end function
function MAIN                                                    ▷ main function
    for i ← 0, n − 1 do
        select[i] = 0
    end for
    rowreduction(cost)
    columnreduction(cost)
    t = row + col
    while allvisited(select) ≠ 1 do
        for i ← 1, n − 1 do
            if select[i] = 0 then
                edgecost[i] = checkbounds(k, i, cost)
            end if
        end for
        min = ∞
        for i ← 1, n − 1 do
            if select[i] = 0 then
                if edgecost[i] < min then
                    min = edgecost[i]
                    k = i
                end if
            end if
        end for
        select[k] = 1
        for p ← 1, n − 1 do
            cost[j][p] = ∞
        end for
        for p ← 1, n − 1 do
            cost[p][k] = ∞
        end for
        cost[k][j] = ∞
        rowreduction(cost)
        columnreduction(cost)
    end while
end function
```

5 Evaluation of Algorithm

The following matrix is the Cost Matrix which shows the distance between the two cities.

$$\text{Cost Matrix} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} - & 10 & 8 & 9 & 7 \\ 10 & - & 10 & 5 & 6 \\ 8 & 10 & - & 8 & 9 \\ 9 & 5 & 8 & - & 6 \\ 7 & 6 & 9 & 6 & - \end{pmatrix} \end{array}$$

We know that the sum of row minimum gives us the lower bound.Now we have to find the reduced matrix by subtracting the minimum element from every row.So,row minimum will be 31.

$$\text{Reduced Matrix} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} - & 3 & 1 & 2 & 0 \\ 5 & - & 5 & 0 & 1 \\ 0 & 2 & - & 0 & 1 \\ 4 & 0 & 3 & - & 1 \\ 1 & 0 & 3 & 0 & - \end{pmatrix} \end{array}$$

In the above reduced matrix there should be a *Zero* in every row and every column.Now apply the column minimum principle which means for a particular city I will have come into that city from other city.So,now subtracting the column minimum , we get the lower bound 31+1 = 32.

$$\text{Reduced Matrix} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} - & 3 & 0^{(2)} & 2 & 0^{(1)} \\ 5 & - & 4 & 0^{(1)} & 1 \\ 0^{(1)} & 2 & - & 0^{(0)} & 1 \\ 4 & 0^{1)} & 2 & - & 1 \\ 1 & 0^{(0)} & 2 & 0^{(0)} & - \end{pmatrix} \end{array}$$

Now we have to make the assignments *Zero's* in the above matrix.If we can not make an assignment at that particular Zero we have to go for the next highest element in that row.If there is same value,we have to calculate the sum of row penalty and column penalty.If the sum of penalties is more then we have to make that assignment because we incur an additional cost if we don't make that assignment.

Now if we don't have path from 1 to 3 i.e,

$$X_{(13)} = 0 \tag{1}$$

we have an additional cost of 2 and the lower bound becomes 32+2 = 34.

If we have path from 1 to 3 i.e,

$$X_{(13)} = 1 \tag{2}$$

.

we eliminate the respective row and column.

$$\text{Reduced Matrix} = \begin{array}{c} \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 4 & 5 \end{array} \\ \begin{pmatrix} 5 & - & 0 & 1 \\ - & 2 & 0 & 1 \\ 4 & 0 & - & 1 \\ 1 & 0 & 0 & - \end{pmatrix} \end{array}$$

We have to check whether every row and every column has a zero otherwise we have to subtract the minimum element from every respctive row or column.So,we get

$$\text{Reduced Matrix} = \begin{array}{c} \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 4 & 5 \end{array} \\ \begin{pmatrix} 4 & - & 0 & 0 \\ - & 2 & 0 & 0 \\ 3 & 0 & - & 0 \\ 0 & 0 & 0 & - \end{pmatrix} \end{array}$$

Here we made a column reduction for two columns(1 & 5)which gives a lower bound of 32+1+1 = 34.

Again we calculate the penalty for the above matrix.

$$\text{Reduced Matrix} = \begin{array}{c} \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 4 & 5 \end{array} \\ \begin{pmatrix} 4 & - & 0^{(0)} & 0^{(0)} \\ - & 2 & 0^{(0)} & 0^{(0)} \\ 3 & 0^{(0)} & - & 0^{(0)} \\ 0^{(3)} & 0^{(0)} & 0^{(0)} & - \end{pmatrix} \end{array}$$

Now if we don't have path from 5 to 1 i.e,

$$X_{(51)} = 0 \tag{3}$$

we have an additional cost of 3 and the lower bound becomes 34+3 = 37.

If we have path from 5 to 1 i.e,

$$X_{(51)} = 1 \tag{4}$$

.

we eliminate the respective row and column.

$$\text{Reduced Matrix} = \begin{array}{c} \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 4 & 5 \end{array} \\ \begin{pmatrix} - & 0 & 0 \\ 2 & 0 & - \\ 0 & - & 0 \end{pmatrix} \end{array}$$

Here we a Zero in every row and column.So,the bound remains the same i.e, 34+0 = 34.

If we calculate the penalty for the above matrix,

$$\text{Reduced Matrix} = \begin{array}{c} \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 4 & 5 \end{array} \\ \begin{pmatrix} - & 0^{(0)} & 0^{(0)} \\ 2 & 0^{(2)} & - \\ 0^{(2)} & - & 0^{(0)} \end{pmatrix} \end{array}$$

Now if we don't have path from 3 to 4 i.e,

$$X_{(34)} = 0 \tag{5}$$

we have an additional cost of 2 and the lower bound becomes 34+2 = 36.

If we have path from 3 to 4 i.e,

$$X_{(34)} = 1 \tag{6}$$

.

we eliminate the respective row and column.

$$\text{Reduced Matrix} = \begin{array}{c} \\ 2 \\ 4 \end{array} \begin{array}{c} \begin{array}{cc} 2 & 5 \end{array} \\ \begin{pmatrix} - & 0 \\ 0 & - \end{pmatrix} \end{array}$$

Here we a Zero in every row and column.So,the bound remains the same i.e, 34+0 = 34.

Now we have only two possible assignments with Zero penalty cost which would give us a feasible solution.

$$X_{(25)} = 1 \tag{7}$$

$$X_{(42)} = 1 \tag{8}$$

On adding all the paths we get,

$$X_{(13)} + X_{(34)} + X_{(42)} + X_{(25)} + x_{(51)} = 34 \tag{9}$$

References

[1] http://en.wikipedia.org/wiki/Travelling_salesman_problem.

[2] http://lcm.csa.iisc.ernet.in/dsa/node187.html.

[3] http://www.youtube.com/watch?v=nN4K8xA8ShM, 2010.