METU, Computer Engineering Fall 2015
CENG567 - Design and Analysis of Algorithms
Homework 3 - Pseudocodes
Erenay DAYANIK - 1819192

1 Backtracking

Algorithm 1 Algorithm for "Travelling Salesman Problem" using backtracking method [1]

```
// converts arg1 to reduced matrix
procedure REDUCE(arg1)
    Input: arg1
    Output: value

    m ← arg1.size();
    value ← 0;
    for each i ∈ [0, m − 1] do
        min ← arg1[i, 0];
        for each j ∈ [1, m − 1] do
            if arg1[i,j] > min then
                min ← arg1[i, j];
            end if
        end for
        for each j ∈ [0, m − 1] do
            arg1[i,j] ← arg1[i,j]- min;
        end for
        value ← value + min;
    end for
    for each j ∈ [0, m − 1] do
        min ← arg1[0, j];
        for each i ∈ [1, m − 1] do
            if arg1[i,j] > min then
                min ← arg1[i, j];
            end if
        end for
        for each i ∈ [0, m − 1] do
            arg1[i,j] ← arg1[i,j]- min;
        end for
        value ← value + min;
    end for
    return value
end procedure
```

```
// computes boundary function
procedure REDUCEBOUND(partialSolution)
    Input: partialSolution
    Output: ans
    GlobalVariables: costMatrix,V
    // V: all vertices in the graph
    // M: cost matrix stores weight of each edge in the graph
    // partialSolution : [x_0, x_1, ..., x_{m-1}]
    if m = V.size() then
        return (COST(partialSolution));
    end if
    M'[0][0] ← ∞;
    for each y ∈V\partialSolution do
        M'[0][j] ← costMatrix[x_{m-1}][y];
        j ← j + 1;
    end for
    i ← 1;
    for each x ∈V\partialSolution do
        M'[i][0] ← costMatrix[x][x_0];
        i ← i + 1;
    end for
    i ← 1;
    for each x ∈V\partialSolution do
        j ← 1;
        for each y ∈V\partialSolution do
            M'[i][j] ← costMatrix[x][y];
            j ← j + 1;
        end for
        i ← i + 1;
    end for
    ans ← REDUCE(M');
    for i = 1; i < m-1; i++  do
        ans ← ans + costMatrix[x_{i-1}][x_i];
    end for
    return ans
end procedure
```

```
procedure BACKTRACKING(level,partialSolution)
    Input: level,partialSolution
    // partialSolution: [x_0, x_1, ..., x_{level-1}];
    Output: OptX,OptC
    GlobalVariables: C_level (level=0,1,2,...,n-1)
    // C_level: choice set for that level
    // n: number of nodes in the graph
    if level = n then
        C ← COST(partialSolution);
        if C < OptC then
            OptC ← C;
            OptX ← partialSolution;
        end if
    end if
    if level = 0 then
        C_level ← {0};
    else if level = 1 then
        C_level ← {1, 2, 3, ...., n − 1};
    else
        C_level ← C_{level−1} \ {x_{level−1}};
    end if
    B ← REDUCEBOUND(partialSolution);
    for each x ∈ C_level do
        if B >= OptC then
            return
        end if
        x_level ← x;
        BACKTRACKING(level + 1, partialSolution);
    end for
end procedure
```

2 Branch & Bound

Algorithm 2 Algorithm for "Travelling Salesman Problem" using branch & bound method [1]

```
procedure BRANCHANDBOUND(level,partialSolution)
    Input: level,partialSolution
    // partialSolution: [x_0, x_1, ..., x_{level-1}];
    Output: OptX,OptC
    GlobalVariables: C_level (level=0,1,2,...,n-1)
    // C_level: choice set for that level
    // n: number of nodes in the graph
    if level = n then
        C ← COST(partialSolution);
        if C < OptC then
            OptC ← C;
            OptX ← partialSolution;
        end if
    end if
    if level = 0 then
        C_level ← {0};
    else if level = 1 then
        C_level ← {1, 2, 3, ...., n − 1};
    else
        C_level ← C_{level−1} \ {x_{level−1}};
    end if
    count ← 0;
    for each x ∈ C_level do
        x_level ← x;
        nextchoice[count] ← x;
        nextbound[count] ← REDUCEBOUND(partialSolution);
        count ← count + 1;
    end for
    // Sort nextchoice and nextbound so that nextbound is an increasing order
    SORT(nextchoice, nextbound);
    for i = 1; i < count-1; i + +  do
        if nextbound[i] >= OptC then
            return
        end if
        x_level ← nextchoice[i];
        BRANCHandBOUND(level + 1, partialSolution);
    end for
end procedure
```

References

[1] Donald L. Kreher , Douglas R. Stinson. *Combinatorial Algorithms*, CRC Press, New York, 2nd edition, 1999.