

1 Backtracking

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

```
1: // converts arg1 to reduced matrix
2: procedure REDUCE(arg1)
3:   Input: arg1
4:   Output: value
5:
6:    $m \leftarrow \text{arg1.size}();$ 
7:    $value \leftarrow 0;$ 
8:   for each  $i \in [0, m - 1]$  do
9:      $min \leftarrow \text{arg1}[i, 0];$ 
10:    for each  $j \in [1, m - 1]$  do
11:      if  $\text{arg1}[i, j] > min$  then
12:         $min \leftarrow \text{arg1}[i, j];$ 
13:      end if
14:    end for
15:    for each  $j \in [0, m - 1]$  do
16:       $\text{arg1}[i, j] \leftarrow \text{arg1}[i, j] - min;$ 
17:    end for
18:     $value \leftarrow value + min;$ 
19:  end for
20:  for each  $j \in [0, m - 1]$  do
21:     $min \leftarrow \text{arg1}[0, j];$ 
22:    for each  $i \in [1, m - 1]$  do
23:      if  $\text{arg1}[i, j] > min$  then
24:         $min \leftarrow \text{arg1}[i, j];$ 
25:      end if
26:    end for
27:    for each  $i \in [0, m - 1]$  do
28:       $\text{arg1}[i, j] \leftarrow \text{arg1}[i, j] - min;$ 
29:    end for
30:     $value \leftarrow value + min;$ 
31:  end for
32:  return  $value$ 
33: end procedure
```

```

34: // computes boundary function
35: procedure REDUCEBOUND(partialSolution)
36:   Input: partialSolution
37:   Output: ans
38:   GlobalVariables: costMatrix, V
39:   // V: all vertices in the graph
40:   // M: cost matrix stores weight of each edge in the graph
41:   // partialSolution :  $[x_0, x_1, \dots, x_{m-1}]$ 
42:   if  $m = V.size()$  then
43:     return  $COST(partialSolution)$ ;
44:   end if
45:    $M'[0][0] \leftarrow \infty$ ;
46:   for each  $y \in V \setminus partialSolution$  do
47:      $M'[0][j] \leftarrow costMatrix[x_{m-1}][y]$ ;
48:      $j \leftarrow j + 1$ ;
49:   end for
50:    $i \leftarrow 1$ ;
51:   for each  $x \in V \setminus partialSolution$  do
52:      $M'[i][0] \leftarrow costMatrix[x][x_0]$ ;
53:      $i \leftarrow i + 1$ ;
54:   end for
55:    $i \leftarrow 1$ ;
56:   for each  $x \in V \setminus partialSolution$  do
57:      $j \leftarrow 1$ ;
58:     for each  $y \in V \setminus partialSolution$  do
59:        $M'[i][j] \leftarrow costMatrix[x][y]$ ;
60:        $j \leftarrow j + 1$ ;
61:     end for
62:      $i \leftarrow i + 1$ ;
63:   end for
64:    $ans \leftarrow REDUCE(M')$ ;
65:   for  $i = 1; i < m-1; i++$  do
66:      $ans \leftarrow ans + costMatrix[x_{i-1}][x_i]$ ;
67:   end for
68:   return ans
69: end procedure
70:
71:
72:
73:
74:
75:

```

```

76: procedure BACKTRACKING(level,partialSolution)
77:   Input: level,partialSolution
78:   // partialSolution:  $[x_0, x_1, \dots, x_{level-1}]$ ;
79:   Output: OptX,OptC
80:   GlobalVariables:  $C_{level}$  ( $level=0,1,2,\dots,n-1$ )
81:   //  $C_{level}$ : choice set for that level
82:   //  $n$ : number of nodes in the graph
83:   if  $level = n$  then
84:      $C \leftarrow COST(partialSolution)$ ;
85:     if  $C < OptC$  then
86:        $OptC \leftarrow C$ ;
87:        $OptX \leftarrow partialSolution$ ;
88:     end if
89:   end if
90:   if  $level = 0$  then
91:      $C_{level} \leftarrow \{0\}$ ;
92:   else if  $level = 1$  then
93:      $C_{level} \leftarrow \{1, 2, 3, \dots, n - 1\}$ ;
94:   else
95:      $C_{level} \leftarrow C_{level-1} \setminus \{x_{level-1}\}$ ;
96:   end if
97:    $B \leftarrow REDUCEBOUND(partialSolution)$ ;
98:   for each  $x \in C_{level}$  do
99:     if  $B \geq OptC$  then
100:       return
101:     end if
102:      $x_{level} \leftarrow x$ ;
103:     BACKTRACKING( $level + 1$ ,partialSolution);
104:   end for
105: end procedure

```

2 Branch & Bound

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

```
1: procedure BRANCHANDBOUND(level,partialSolution)
2:   Input: level,partialSolution
3:   // partialSolution:  $[x_0, x_1, \dots, x_{level-1}]$ ;
4:   Output: OptX, OptC
5:   GlobalVariables:  $C_{level}$  ( $level=0,1,2,\dots,n-1$ )
6:   //  $C_{level}$ : choice set for that level
7:   //  $n$ : number of nodes in the graph
8:   if  $level = n$  then
9:      $C \leftarrow COST(partialSolution)$ ;
10:    if  $C < OptC$  then
11:       $OptC \leftarrow C$ ;
12:       $OptX \leftarrow partialSolution$ ;
13:    end if
14:  end if
15:  if  $level = 0$  then
16:     $C_{level} \leftarrow \{0\}$ ;
17:  else if  $level = 1$  then
18:     $C_{level} \leftarrow \{1, 2, 3, \dots, n - 1\}$ ;
19:  else
20:     $C_{level} \leftarrow C_{level-1} \setminus \{x_{level-1}\}$ ;
21:  end if
22:   $count \leftarrow 0$ ;
23:  for each  $x \in C_{level}$  do
24:     $x_{level} \leftarrow x$ ;
25:     $nextchoice[count] \leftarrow x$ ;
26:     $nextbound[count] \leftarrow REDUCEBOUND(partialSolution)$ ;
27:     $count \leftarrow count + 1$ ;
28:  end for
29:  // Sort nextchoice and nextbound so that nextbound is an increasing order
30:   $SORT(nextchoice, nextbound)$ ;
31:  for  $i = 1; i < count-1; i++$  do
32:    if  $nextbound[i] \geq OptC$  then
33:      return
34:    end if
35:     $x_{level} \leftarrow nextchoice[i]$ ;
36:     $BRANCHandBOUND(level + 1, partialSolution)$ ;
37:  end for
38: end procedure
```

References

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