Search Algorithms

Apoorva Mahishwar (519795), Mohamed Benchat (516589)

May 31, 2022

1 Permute & Search

The following algorithm is based on using the permutation table for each truck to create a tree-like structure that includes the nodes from a given graph as keys and the sorted adjacent nodes (of each node) as values. The search algorithm then starts from the origin, visits the first adjacent node of each key until reaching the destination.

Algorithm 1 Sort list using permutation then run search of route

```
1:  procedure SortList(List, Permutation)
2:      for element ∈ List do
3:          sortedList ← SwapPosition(element, Permutation).  ▷ Swap position of element using Permutation
4:      return sortedList
5:
6:  procedure CreateStructure(Graph, Permutation)                ▷ Rebuild graph
7:      for node ∈ Graph do
8:          Adjacents ← ListOfAdjacentNodes(node)
9:          SortedList ← SortList(Adjacents, Permutation)
10:         Reconstruction ← { node : SortedList }               ▷ Append not assignment!
11:     return Reconstruction
12:
13: procedure CreateRoute(Origin, Destination, Graph, Permutation)
14:     Route ← Origin                                           ▷ Append not assignment!
15:     Reconstruction ← CreateStructure(Graph, Permutation)
16:     while Destination ∉ Route do
17:         Adjacents ← GetElements(Reconstruction, Origin)
18:         for Adjacent ∈ Adjacents do
19:             if Adjacent ∉ Route then
20:                 Route ← Adjacent                             ▷ Append not assignment!
21:                 Origin ← Adjacent
22:                 Break
23:     return Route
```

2 Embed & Search

Using the permutation table for each truck to embed the order as weight of the respective node. The search algorithm then starts from the origin, visits the adjacent node with the minimum weight, take this node as the new origin, this is repeated until reaching the destination.

Algorithm 2 Embed permutation order then run search of route

```
1: procedure BuildWeightsStructure(P, OD)
2:     SETS:
3:     P : Permutation Table
4:     OD : Pair of origin and destination nodes
5:     W : Weights Table
6:     S_A : Set of adjacent nodes
7:     R : Route
8:     PARAMETERS:
9:     O : Origin
10:    D : Destination
11:    N : Current Node
12:    for P_i ∈ P do
13:        O ← GetOrigin(OD)
14:        W_i(O) ← 0
15:        for N ∈ P_i do
16:            if N ≠O then
17:                W_i(N) ← Index(N, P_i)
18:    return W
19:
20: procedure SearchWeightsStructure(G, P, OD)
21:    SETS:
22:    G : Graph
23:    P : Permutation Table
24:    OD : Pair of origin and destination nodes
25:    W : Weights Table
26:    S_A : Set of adjacent nodes
27:    R : Route
28:    PARAMETERS:
29:    O : Origin
30:    D : Destination
31:    N : Current Node
32:    for W_i ∈ W do
33:        O ← GetOrigin(OD)
34:        D ← GetDestination(OD)
35:        N ← O
36:        R_i ← O                                  ▷ Append not assignment!
37:        while D ∉ R_i do
38:            S_A ← Adjacents(N, G)                ▷ Get adjacent nodes of N from Graph G
39:            if D ∈ S_A then
40:                R_i ← D
41:                Break
42:            if Intersection(S_A, R_i) ≠ ∅ then
43:                Remove Intersection(S_A, R_i) from S_A
44:            N ← GetKey(min(W_i(S_A)))
45:            R_i ← N                              ▷ Append not assignment!
46:    return R
```

3 Conclusion

We present the aforementioned methods in a simple pseudo-code that can be directly interpreted in a high-level programming language such as *Python*. We intentionally skipped few details (e.g: position swapping, index extraction, key extraction ...) in order to keep the pseudo-code as simple as possible for the readers.