Talbi 1.11

Algorithm CVRP

```
Input: A graph G = (V,A), the index of the depot i, and the capacity of the trucks Q
Output: A set of routes R with minimum cost
R = {}
T = {} //partial solution
//Next state generator and selection
Loop
   T.push(V[i]) //push the depot on first since all vehicles must start there
   Remove V[i] from V
   Find edge (Vi, Vj) with minimum cost from last node in T, Vi
   //Feasibility
   If demand of Vj <= remaining capacity Q of Vi
      T.push(Vj) as next stop
   //Solution
   If V is empty
      R = T
      Break loop
End Loop
Return R
```

Talbi 1.17

In [4] the objective function is discussed and it can be noted that although the objective function is not computationally difficult to compute or derive, it is very memory intensive since it must be saved at each step of the search process. This will be compounded with larger problems becoming memory inefficient for even moderately large examples.

b) (i)

Problem Domain

The NPC problem which will be considered is a variation of the VRP where we have some number of UAVs D, each with a uniform amount of fuel F, and some number of target locations which need to be observed T. The base from which they leave will be the “depot” in this case and will be a special element of T known as T_{Home}. The cost c of the edge (t_i, t_j) will be the amount of fuel required to travel that distance. This is similar to what was done in [1] without the consideration of the complex urban environments. The constraints will be that each location must only be visited by one drone, since it is unnecessary to have multiple viewing the same location. We will assume that the terrain is level and all drones fly at the same altitude. More formally, the problem can be stated as follows:

Given a graph G = (T, C), where T is the set of targets and C is the set of edge costs in gallons of fuel, find a set
$r = \{T_{Home} \cup t_i \ \forall i \in T | \ c < F \ and \ c \ is \ minimized\}$ for each drone in D.
Additionally $\forall d_i \in D, \ r_i \cap r_j = \{T_{home}\} \forall i \neq j$.
The output will be the set $R = \{r_1, \ r_2, ..., r_i\} \forall d_i \in D$
As described in [2] and [3] the vehicle routing problem and its variants are NP-Hard.

(ii) **Algorithm Domain Specification**

- *name: Global-Search-bfs* $(P_i, \ P_o)$*; gs-bfs*
- *domains:* P_i *is set-of-candidates,* P_o *are the sets of solutions,* P_p *is set of partial solutions of "generated nodes"*
  - ***next-state-generator***

    i) ***selection*** *of a partial solution* y *in* P_p *based upon its superiority and put in* P_c *and delete from* P_p

    ii) ***Generation*** *of* all *next states* x_j *of* y
  - ***feasibility*** $(x_j \ , \ y) \rightarrow$ *boolean [if true union (xj , y) and put result in* P_p*]*
  - ***solution*** $(y) \rightarrow$ *boolean; z = y; delay termination and find all "optimal" solutions (if satisfizing accept one/first solution)*
  - *objective solution(*P_p*)* $\rightarrow$ *"ordered set/well founded set over* P_p*"*

PD/AD integration

-Set of candidates: A list of reachable targets $t_i \in T$ for drone $d_i \in D$
-Next-state generator: Generate a list G of the next reachable target $t_i \in T$ for each drone $d_i \in D$
-Selection function: Select next reachable target $t_i \in T$ for a drone $d_i \in D$ such that the fuel cost is minimized.
-Feasibility function: If the drone d_i has enough fuel to make it back to T_{home} after flying to t_i then it is a feasible choice
-Solution function: If for each $t_i \in T$, t_i has been visited by a drone and for each $d_i \in D$, d_i has returned to T_{home} then it is a potential solution
-Objective function: The total amount of fuel used so far in gallons
-Heuristic: The fuel cost to get every drone from their current location back to the T_{home}
Admissibility of Heuristic: In the final goal state all drones will be back at T_{home} and the list of available targets will be empty. Therefore at one search state before the goal the highest possible cost will be the cost to get all drones back to T_{home} and will never overestimate the cost to the goal.

Heuristic time complexity: This heuristic will require O(n) time where n is the number of drones in D since only one computation is required for each.

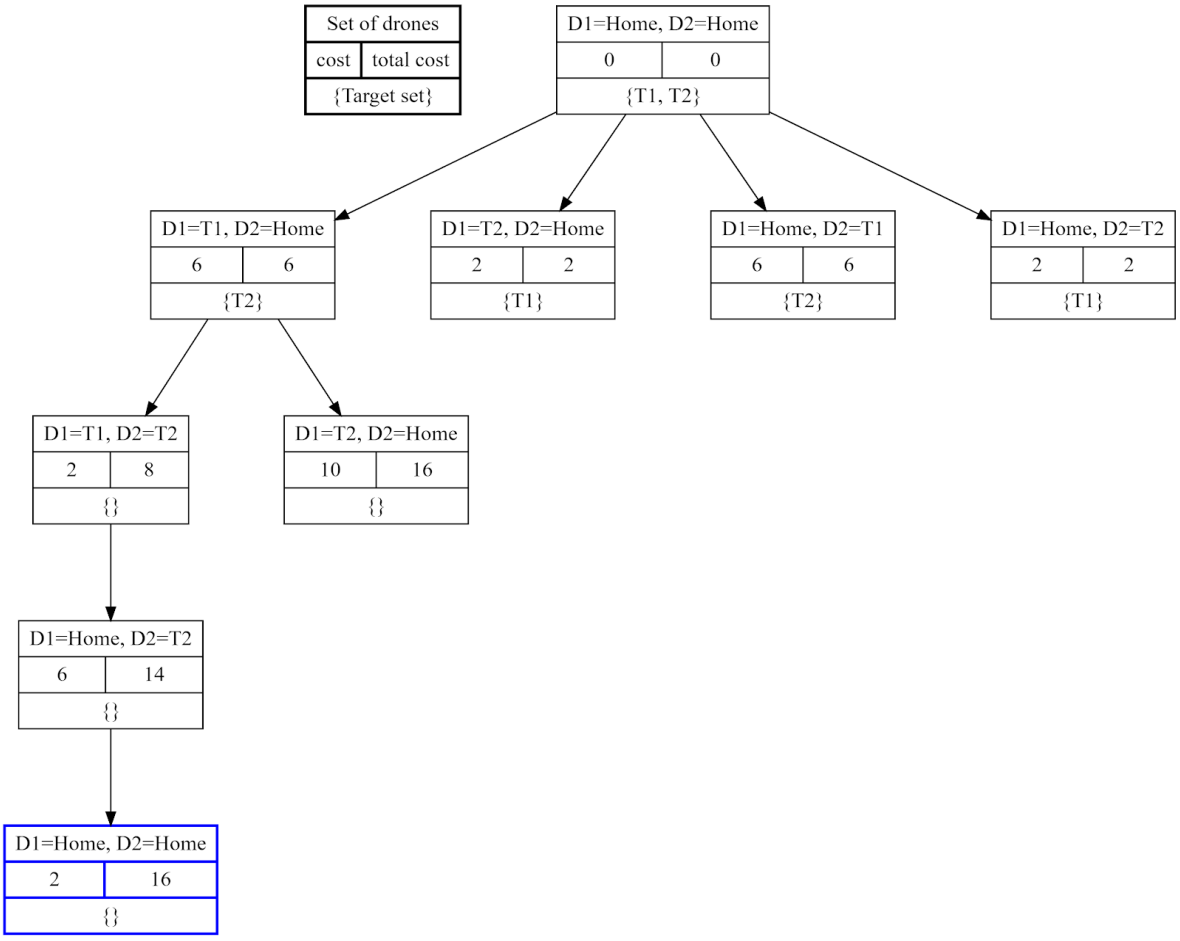
(iii) **Functional Pseudocode**

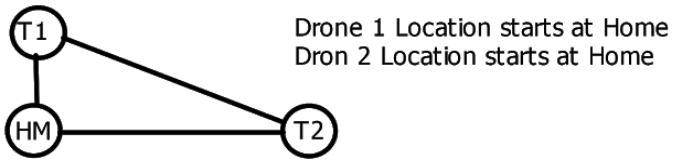
Algorithm VRP A*

```
struct searchNode
   Let D be a map of drones to their current location
   List T of targets left to visit
   Current total cost C

Let goal be a searchNode such that
   For all d in D
      d.currentLocation == home
   T is empty

function reconstruct_path(cameFrom, current)
   total_path = {current}
   Loop while current in cameFrom.Keys:
      current = cameFrom[current]
      total_path.prepend(current)
   End loop
   return total_path

//heuristic
function h(searchNode)
   heuristicCost = 0
   for d in searchNode.D
      heuristicCost += cost(d.currentLocation, home)
   return heuristicCost

//next state generator
function generateNeighbors(searchNode)
   Let N be a list of searchNodes
   for d in searchNode.D
      for t in searchNode.T
         create newSearchNode
         newSearchNode.D[d] = t
         N.add(newSearchNode)
   Return N

function A_Star(start, goal, h)
   Let openSet be a priorityQueue
   openSet = {start}
```

```
    Let cameFrom be a map from a search node n to the node immediately preceding it on the cheapest path from
start to n currently known
    cameFrom = an empty map

    Let G be a map from search node n to the cost of the cheapest path from start to n currently known
    Initialize all values of G to infinity
    G[start] = 0

    Let F be map from a search node n to an int G[n] + h(n)
    Initialize all values of F to infinity
    F[start] = h(start)

    Loop while openSet is not empty
        current = the node in openSet having the lowest F[] value
//solution
        if current == goal
            return getPath(cameFrom, current)

        openSet.Remove(current)

        generateNeighbors(current)
        for each neighbor of current
            // cost from start to the neighbor through current
            tentative_G = G[current] + cost(current, neighbor)
            if tentative_G < G[neighbor]
                cameFrom[neighbor] = current
                G[neighbor] = tentative_G
                F[neighbor] = G[neighbor] + h(neighbor)
                if neighbor not in openSet
                    openSet.add(neighbor)
    End loop
    return failure
```

(iv) The following image is a partial search graph showing some of the search states of the simple example shown below.





(v) We will compare the heuristic discussed in this assignment with the one described in [5]. First and foremost, the heuristic used here is specific to the MVRP while the one in [5] is a general heuristic meant to be used with any variation of the VRP. The time complexity required for our heuristic is O(n) where n is the number of drones being considered since there is only computation required for each. The complexity of the heuristic framework in [5], although based on the same framework, is changed slightly

for each variant of the problem and therefore may have slightly different complexity requirements. The advantage is that it can be quickly adapted. The results showed that this framework could become the standard for VRP since it showed the same or better performance for all variants of the VRP compared to the state of the art. The heuristic shown here may perform better on this specific example but it is a far cry from a general framework for solving VRP problems.

References

[1]Semsch, E., Jakob, M., Pavlicek, D., & Pechoucek, M. (2009, September). Autonomous UAV surveillance in complex urban environments. In *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology* (Vol. 2, pp. 82-85). IEEE.

[2]Lenstra, J. K., & Kan, A. R. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, *11*(2), 221-227.

[3]https://en.wikipedia.org/wiki/Vehicle_routing_problem#:~:text=The%20vehicle%20routing%20problem%20(VRP,travelling%20salesman%20problem%20(TSP)

[4]Laporte, G. (1992). The vehicle routing problem: An overview of exact and approximate algorithms. *European journal of operational research*, *59*(3), 345-358.

[5]Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & operations research*, *34*(8), 2403-2435.