Project Phase 2
Vincent Bownes

Problem Domain
Hybrid algorithm for UAV routing and supply delivery.
The first 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, and some locations will be marked as "high priority" and must be visited first. 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 $r = \{T_{Home} \cup t_i\ \forall i \in T | c < F \text{ and } c \text{ is minimized}\}$ for each drone in D. Additionally $\forall d_i \in D,\ r_i \cap r_j = \{T_{home}\} \forall i \neq j$.

The second NPC problem to be incorporated will be the 3D bin packing problem [2]. Assume that all locations require supply drops of varying size and each drone has a fixed 3 dimensional capacity L. We want to minimize the number of drones required to make supply drops. This problem on its own can be stated as follows [3]: Given a set of items I, a set of sizes S representing the size of each item and a bin capacity L find the minimum K such that $K = \sum_{j=1}^{n} y_j$ where $y_j = 1$ if drone d_j is used to carry supplies.

The combined objective function can be formulated as follows:
$H = [cost(t_i, t_j)\ \forall i \in r_x\ \forall x \in D] + \sum_{j=1}^{n} y_j$. Our goal is to minimize this hybrid objective function. In other words find the routes that use the least amount of fuel while simultaneously finding the least number of drones required to make all supply drops.

D_i: G = (T, C), where T is the set of targets and C is the set of edge costs in gallons of fuel and each node T has a set of items I with sizes S that must be dropped there
D_o: A map M from targets to drones which represents which drone visited which targets and all targets have drones assigned to them

Algorithm Domain Selection and Specification

name: Global-Search Breath First Search (Di, Do); gs-bfs

domains: Di is set-of-candidates, Do are sets of solutions (solution space of subsets)

operations:

***I*(*n*)**: *n* is the start node, {} which is initially placed on the OPEN list
***state**:* set of nodes explored (OPEN) and set of nodes expanded (CLOSED)
set of candidates: nodes on the frontier grouped by inclusion in the OPEN list
selection function: choose node n' with smallest $f(n')$, place it on OPEN list and its descendants on the CLOSED list
feasibility function: a node is feasible if $\forall d_i \in D,\ r_i \cap r_j = \{T_{home}\}\ \forall i \neq j$ meaning that all drones start and end at the depot. AND $\forall d_i \in D$ the remaining fuel in $d_i \geq$ the cost from the current node to the depot. AND the sum of the sizes S of items I packed into any d_i is less than the capacity L.
***solution function**:* a node is a solution (terminal) if all targets in M_t have a drone mapped to them AND all drones in M_d are back at the depot T_{home}. Terminal nodes are placed on the CLOSED list.

A* [4] will be used for this hybrid problem as the deterministic approach since a heuristic should be beneficial in pruning the search tree as opposed to a naive greedy algorithm. A BFS [5] approach is taken as opposed to a DFS [6] approach because due to the constraints from the bin packing problem we want to use as few drones as possible and the deeper down the search tree we go the more drones will be added. Therefore the ideal solution will be closer to the root and we would want to search all nodes at each level before going any deeper.

Algorithm Design and Refinement

name: Global-Search-bfs (Di, Do); gs-bfs

domains: Di is set-of-candidates, Do are the sets of solutions, Dp is set of partial solutions of “generated nodes”- setofsets, boolean
imports: A map from targets to drones and drones to locations. Lists for the open and closed lists
operations:

I(x); x in D_i

O(x,z); x in D_i, z in D_o;

“condition on z being an optimal (satisfizing) solution”

I’(x,y); x in Di, y in Dp; condition on y being a partial solution in Dp

Dp is the “open” list; Dc is the ”closed” list

State is defined by two maps M_d from drones to their current locations and M_t from targets to the drone that has visited them (or none)

Next-state-generator

i) selection Use the function f(n) = g(n) + h(n) to find the node in the open list with the lowest f(n)

ii) Generation the neighbors of n are generated by moving from each drone in map M_d to one location for each different neighbor and updating M_t accordingly

feasibility (n) $\rightarrow$ a node is feasible if $\forall d_i \in D,\ r_i \cap r_j = \{T_{home}\}\ \forall i \neq j$ meaning that all drones start and end at the depot. AND $\forall d_i \in D$ the remaining fuel in $d_i \geq$ the cost from the current node to the depot. AND the sum of the sizes S of items I packed into any d_i is less than the capacity L.

solution (n) $\rightarrow$ a node is a solution (terminal) if all targets in M_t have a drone mapped to them AND all drones in M_d are back at the depot T_{home}.

heuristics

First we will discuss the h(n) function: h(n) = 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. The pseudocode that follows will be a slightly altered version of the pseudocode submitted for HW6 since it is the same NPC problem.

Hybrid A*

---

struct searchNode
  Let M_d be a map of drones to their current location
  Let M_t be a map of targets to the drone that has visited them
  Current total cost C

Let **goal** be a searchNode such that
  For all d in M_d
    d.currentLocation == home
  For all t in M_t
    t has a location

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 list
  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
```

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]Martello, S., Pisinger, D., & Vigo, D. (2000). The three-dimensional bin packing problem. *Operations research*, *48*(2), 256-267.

[3]https://en.wikipedia.org/wiki/Bin_packing_problem#Formal_statement

[4]https://en.wikipedia.org/wiki/A*_search_algorithm

[5]https://en.wikipedia.org/wiki/Breadth-first_search

[6]https://en.wikipedia.org/wiki/Depth-first_search