Mark Demore, 2d Lt

CSCE686 – Dr. Lamont

Project - Deterministic Design

**PROBLEM:**

Let *G* = (*V*, *A*) be a graph where *V* = {1, …, *n*} is a set of vertices representing air drop locations with the airfield located at vertex 1, and *A* is the set of arcs between them. Every arc (*i, j*) *i* =/=*j* is associated a non-negative cost matrix *C* = (*cij*). Let each vertex be assigned a set *S*,of *n* items, each with a weight *wi* and a value *vi*. Let P be a set of planes, each with a maximum cargo weight *Wi*. Constraints: (i) each air drop in *V*\{1} is visited exactly once by exactly one plane; (ii) all flight plans start and end at the airfield; (iii) . Objective: maximize total value and minimize total cost across all flight plans; *max* , *min*

**Problem Domain Requirements Specification form:**

- domains, D

input Di - Graph G(X,Γ), X:locations. Γ: weighted vertex link set (cost); Set S(W,V), W: item weight, V: item value

output Do – Set of sets(R,L), R:route for each plane, L: load for each plane

- I(x); input conditions on input domain satisﬁed; x in X, link in Γ, set S

- O(x,z); output conditions on output/input domain satisﬁed; i.e.,

a feasible/optimal solution with respect to the input domain   
-- all x assigned  
-- max V (total value)

-- no *wp >* W (max weight)  
-- min C (total cost)

**Problem Domain/Algorithm Domain Integration Specification**

* **Basic search constructs** for A\*

* *next-state-generator* (Di) − > x in X; I(x)
* *selection* (Di) − > x; x in X
* *feasibility* (x, Dp) − > boolean
* *solution* O(x,z) “maximal “; (Dp) − > boolean; z = Dp, i.e., can no longer   
   augment S with an x in X;
* *objective (*Dp*) ->* Do *optimal assignment of all locations*
* imports: ADT( set, set-of-sets):Di Dp Do; Boolean; integer

***algorithm domain requirements speciﬁcation form:***

*• name: A\* (Di, Do)*

*• domains: Di is set-of-candidates,*

*Do are sets of solutions (solution space of subsets)*

*• operations:*

*I(x); x in Di; x is a possible candidate from input set*

*O(x, z); x in Di, z in Do; z is a satisfying solution*

***algorithm domain design speciﬁcation form:***

*• name: A\* (Di, Do)*

*• domains: Di is set-of-candidates, Do are the sets of solutions, Dp is set of partial solutions (one plane’s route and load)*

*• imports: ADT set, list, queue, real/integer/character*

*• operations:*1

*I(x); x in Di*

*O(x, z); x in Di, z in Do;*

*“condition on z being a satisfying 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*

**–** *deﬁne state*

**– *next-state-generator***

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

*“based upon heuristic cost function”*

*ii)* ***Generation*** *of all* *next states xj of y*

**– *feasibility*** *(xj, y) − > boolean [if true union (xj , y) and put result* *in* *Dp]*

**– *solution*** *(y) − > boolean; z = y; delay termination and ﬁnd all*

*“optimal” solutions (if satisfying accept first solution)*

**–** *objective solution (Dp) − > “ordered set over Dp”*

**– *heuristics*** *come from problem domain insight:*

***-- Attempt use PD next state generator to reduce set-of candidates* *ASAP***

***-- Attempt to generate a combination once and only once in combinatorial problem domain***

***-- Attempt to generate early pruning condition simple solution check***

***algorithm domain intermediate speciﬁcation form: (iterative)***

*• Heuristics: distance to next airdrop location, value of load item added*

*• Data structures: input – graph: set of nodes (locations), set of edge weight (cost between each location), set of items weight and value, set of planes with max weight; output – list of sets (route for each plane, and load for each plane)*

***algorithm domain function speciﬁcation form: (iterative)***

*•*  Function A\*(initial, Expand, Goal, Cost, Heuristic)

q <- New-Priority-Queue()

Insert (initial, q, Heuristic(initial))

**while** q is not empty

**do** current <- Extract-Min(q)

**if** Goal(current) then **return** solution

**for** each next in Expand(current)

**do** Insert (next, q, Cost(next) + Heuristic(next))

return failure

**References**

[1] [https://en.wikipedia.org/wiki/Knapsack\_problem](https://en.wikipedia.org/wiki/Knapsack_problem#:~:text=The%20knapsack%20problem%20is%20a,is%20as%20large%20as%20possible.)

[2] <https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/>

[3] <https://medium.com/@fabianterh/how-to-solve-the-knapsack-problem-with-dynamic-programming-eb88c706d3cf>

[4] <https://en.wikipedia.org/wiki/Vehicle_routing_problem>

[5] <https://developers.google.com/optimization/routing/vrp>

[6] <https://bib.irb.hr/datoteka/433524.Vehnicle_Routing_Problem.pdf>