

# Algorithmic Design for a Doubly NP-Complete Problem

Mark Demore, 2d Lt

## I. INTRODUCTION

**T**HE United States Air Force core mission set includes the transportation of troops, supplies and other assets. This task presents a complex logistics problem. Namely, what is the optimal load for each aircraft, and what is the optimal route for each aircraft to take? These questions are remarkably similar to the knapsack and vehicle routing problems, two notorious NP-Complete problems. An efficient algorithm which can solve this dilemma in tandem would be of great benefit to the USAF. Both a deterministic and stochastic approach to developing an algorithm would be insightful. A clear definition of this problem and two potential algorithmic solutions follow.

## II. PROBLEM STATEMENT

A real world problem that incorporates two different NP-Complete problem models is a airlift scheduling program for Air Mobility Command. While there are certainly many other considerations in the actual application of such a program, it can be simplified to a combination of the Knapsack problem and the Vehicle Routing problem. For example, the knapsack problem can be used to determine how to load the planes, maximizing the value of cargo that can fit in each aircraft, and the vehicle routing problem can be used to determine the order in which to drop off the cargo.

The Knapsack Problem can be defined as:

Let  $S$  be a set of  $n$  items, each with a weight  $w_i$  and a value  $v_i$ , and paired with a maximum cargo weight  $W$ . Constraints:  $\sum_{i=1}^n w_i \leq W$ . Objective: maximize value.

or:

$$\begin{aligned} &\text{maximize } \sum_{i=1}^n v_i x_i \\ &\text{subject to } \sum_{i=1}^n w_i x_i \leq W \\ &\text{and } x_i \in \{0, 1\} \end{aligned}$$

The Vehicle Routing Problem can be defined as:

Let  $G = (V, A)$  be a graph where  $V = \{1, \dots, n\}$  is a set of vertices representing air drop locations with the airfield located at vertex 1, and  $A$  is the set of arcs. With every arc  $(i, j)$   $i \neq j$  is associated a non-negative cost matrix  $C = (c_{ij})$ . Constraints: (i) each air drop in  $V \setminus \{1\}$  is visited exactly once by exactly one plane; (ii) all flight plans start and end at the airfield. Objective: minimize cost.

or:

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}$$

subject to:

$$\sum_{i \in V} x_{ij} = 1 \quad \forall j \in V \setminus \{0\}$$

$$\begin{aligned} &\sum_{j \in V} x_{ij} = 1 \quad \forall i \in V \setminus \{0\} \\ &\sum_{i \in V} x_{i0} = K \\ &\sum_{j \in V} x_{0j} = K \\ &\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq r(S), \quad \forall S \subseteq V \setminus \{0\}, S \neq \emptyset \\ &x_{ij} \in \{0, 1\} \quad \forall i, j \in V \end{aligned}$$

These problems can be combined in this context as:

Let  $G = (V, A)$  be a graph where  $V = \{1, \dots, n\}$  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  $(i, j)$   $i \neq j$  is associated a non-negative cost matrix  $C = (c_{ij})$ . Let each vertex be assigned a set  $S$ , of  $n$  items, each with a weight  $w_i$  and a value  $v_i$ . Let  $P$  be a set of planes, each with a maximum cargo weight  $W_i$ . Constraints: (i) each air drop in  $V \setminus \{1\}$  is visited exactly once by exactly one plane; (ii) all flight plans start and end at the airfield; (iii)  $\sum_{i=1}^n w_i \leq W_P$ . Objective: maximize total value and minimize total cost across all flight plans.

The search landscape for these problems entail computing the knapsack problem and the vehicle routing problem for each aircraft, for each combination of airdrop locations and of items at each location. A heuristic will need to be implemented to determine the tradeoff balance between the two objectives of minimizing cost and maximizing value and also for prioritizing which planes (if they are of varying cargo capacity) should be planned first.

$D_i$ :  $G$  = graph of vertices of air drop locations and arcs,  $C$  = cost matrix for each arc,  $P$  = set of available aircraft,  $S$  = set of items needed at each location

$D_o$ :  $R$  = set of sets of routes for each plane,  $L$  = set of sets of loads for each plane

## III. DETERMINISTIC ALGORITHM DESIGN

### Problem Domain Requirements Specification form:

- domains,  $D$

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

output  $D_o$  - 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  $\Gamma$ , 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  $w_p > W$  (max weight)  
– min  $C$  (total cost)

### Problem Domain/Algorithm Domain Integration Specification

- **Basic search constructs** for  $A^*$ 
  - *next-state-generator*  $(D_i) \rightarrow x \text{ in } X; I(x)$
  - *selection*  $(D_i) \rightarrow x; x \text{ in } X$
  - *feasibility*  $(x, D_p) \rightarrow \text{boolean}$
  - *solution*  $O(x,z)$  “maximal “;  $(D_p) \rightarrow \text{boolean}; z = D_p$ ,  
i.e., can no longer  
augment  $S$  with an  $x$  in  $X$ ;
  - *objective*  $(D_p) \rightarrow D_o$  *optimal assignment of all locations*
- imports: ADT( set, set-of-sets): $D_i D_p D_o$ ; Boolean; integer

#### algorithm domain requirements specification form:

- name:  $A^* (D_i, D_o)$
- domains:  $D_i$  is set-of-candidates,  
 $D_o$  are sets of solutions (solution space of subsets)
- operations:  
 $I(x); x \text{ in } D_i; x$  is a possible candidate from input set  
 $O(x, z); x \text{ in } D_i, z \text{ in } D_o; z$  is a satisfying solution

#### algorithm domain design specification form:

- name:  $A^* (D_i, D_o)$
- domains:  $D_i$  is set-of-candidates,  $D_o$  are the sets of solutions,  $D_p$  is set of partial solutions (one plane's route and load)
- imports: ADT set, list, queue, real/integer/character
- operations:1  
 $I(x); x \text{ in } D_i$   
 $O(x, z); x \text{ in } D_i, z \text{ in } D_o$ ;  
“condition on  $z$  being a satisfying solution”  
 $I'(x, y); x \text{ in } D_i, y \text{ in } D_p$ ; condition on  $y$  being a partial solution in  $D_p$   
 $D_p$  is the “open” list;  $D_c$  is the “closed” list  
– define state  
– *next-state-generator* selection of a partial solution  $y$  in  $D_p$  based upon its superiority and put in  $D_c$  and delete from  $D_p$   
“based upon heuristic cost function”  
ii) **Generation** of all next states  $x_j$  of  $y$   
– *feasibility*  $(x_j, y) \rightarrow \text{boolean}$  [if true union  $(x_j, y)$  and put result in  $D_p$ ]  
– *solution*  $(y) \rightarrow \text{boolean}; z = y$ ; delay termination and find all

“optimal” solutions (if satisfying accept first solution)  
– objective solution  $(D_p) \rightarrow$  “ordered set over  $D_p$ ”  
– *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)  
 $q$  j- New-Priority-Queue()  
Insert (initial,  $q$ , Heuristic(initial))  
**while**  $q$  is not empty  
  **do** current j- 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

## IV. STOCHASTIC ALGORITHM DESIGN

#### Problem Domain Requirements Specification form:

- domains,  $D$
- input  $D_i$  – Graph  $G(X, \Gamma)$ ,  $X$ : locations.  $\Gamma$  : weighted vertex link set (cost); Set  $S(W, V)$ ,  $W$ : item weight,  $V$ : item value
- output  $D_o$  – 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  $\Gamma$ , 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  $w_p > W$  (max weight)  
– min  $C$  (total cost)

#### Algorithm domain requirements specification form:

- Name: stochastic-search genetic algorithm
- Domains: Ds is a set of satisfying solutions-a population; the population size n is the cardinality of Ds
- Operations:

I(x); x in Ds; x is a possible solution from population

O(x,z); x in Ds, z in Ds; z is a satisfying solution

**Algorithm domain design specification form:**

- Name: stochastic-search-ga(Ds)
- Domains: Di is set of algorithm-internal solutions, Ds is a set of satisfying solutions
- Imports: ADT set, list, real/integer/character
- Initialization of feasible solutions  $\rightarrow$  Ds; Di empty
- Operations I(x); x in Ds  
O(x,z); x in Ds, z in Ds; condition on z being a satisfying solution
  - Next-solution-generator  $\rightarrow$  x for x in Ds, Ds
    - \* Recombination(crossover) xwith crossover probability
    - \* Mutation xwith mutation probability
    - \* Feasibility(y)  $\rightarrow$  Boolean [if true union(y,Di)] ‘genotype’
  - Fitness/objective function mapping f(x) of each x in Di ‘phenotype’
  - Selection Di  $\rightarrow$  Ds using f(x) as criteria, x in Di
- Axioms:

**Algorithm domain function specification form:**

- Function stochastic-search-ga(Ds)
- Initial condition: generate feasible Dinitial  $\rightarrow$  Ds, Di empty, pc, pm
- Body
  - \* While not time/generation termination do ss-ga loop:
    - Next-state solution/population Ds, Ds; do for each x in Ds, size n
    - Crossover(x) = y with pc
    - Mutation(x) = y with pm
    - If feasibility(y) then union (y, Di)  $\rightarrow$  Di
    - Fitness calculation f(x) for each x in Di
    - Selection(di)  $\rightarrow$  Ds based upon f(x), x in Di
  - \* End ss-ga while loop
  - \* Find optimal z in Ds
  - \* END function

**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 ss-ga(initial, Expand, Goal, Cost, Heuristic, crossover, mutation)
  - q j- New-Priority-Queue()
  - Insert (initial, q, Heuristic(initial))
  - while** generation limit not reached
    - do** current j- Extract-Min(q)
  - crossover(x) = y with pc
  - mutation(x) = y with pm
    - 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)
- [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](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](https://bib.irb.hr/datoteka/433524.Vehnicle_Routing_Problem.pdf)
- [7] <https://www.geophysik.uni-muenchen.de/igel/downloads/inviigenetic.pdf>
- [8] <https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3>
- [9] [https://en.wikipedia.org/wiki/Genetic\\_algorithm](https://en.wikipedia.org/wiki/Genetic_algorithm)
- [10] <http://www.cs.ucc.ie/~dgb/courses/tai/notes/handout12.pdf>
- [11] <https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html>