Algorithmic Design for a Doubly NP-Complete Problem

Mark Demore, 2d Lt

I. Introduction

THE 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:

maximize $\sum_{i=1}^{n} v_i x_i$

subject to $\sum_{i=1}^{n} w_i x_i \leq W$

and $x_i \in \{0,1\} x_i \in \{0,1\}$

The Vehicle Routing Problem can be defined as:

Let *G* = (*V*, *A*) be a graph where $V = \{ 1, \ldots, n\}$ 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 =/=j* is associated a non-negative cost matrix *C* = (c_{ij}). Constraints: (i) each air drop in $Vn\ \{\ 1\}$ 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:

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

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

$$\sum_{i\in V} x_{i0} = K$$

$$\sum_{j\in V} x_{0j} = K$$

$$\sum_{i\notin S} \sum_{j\in S} x_{ij} \geq r(S), \quad \forall S \subseteq V \setminus \{0\}, S \neq \emptyset$$

$$x_{ij} \in \{0,1\} \quad \forall i,j \in V$$

These problems can be combined in this context as:

Let *G* = (*V*, *A*) be a graph where $V = \{ 1, \ldots, n\}$ 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* = (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 $Vn\ \{\ 1\}$ 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,Γ), X:locations. Γ : 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 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 $w_p > W$ (max weight)
– min C (total cost)

Problem Domain/Algorithm Domain Integration Specification

- **Basic search constructs** for A∗
  - *next-state-generator* (D_i) →x in X; I(x)
  - *selection* (D_i) →x; x in X
  - *feasibility* (x, D_p) →boolean
  - *solution* O(x,z) "maximal "; (D_p) →boolean; z = D_p, i.e., can no longer augment S with an x in X;
  - *objective* (D_p) →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∗ (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

The deterministic algorithm is more likely to find an optimal solution to the complex problem, assuming the heuristic does not overestimate but still encompasses all measures of optimality. The downside of the determinsitic algorithm is that the computational complexity of the problem space will require more time and memory than a potential stochastic implementation. Depending on the objective, a speedy, potentially suboptimal solution, or a longer, optimal solution, one implementation may be preferred over the other.

IV. Stochastic Algorithm Design

Problem Domain Requirements Specification form:

- domains, D

input D_i - Graph G(X,Γ), X:locations. Γ : 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 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 $w_p > \text{W}$ (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 →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 →x for x in Ds, Ds
    * Recombination(crossover) xwith crossover probability
    * Mutation xwith mutation probability
    * Feasibility(y) →Boolean [if true union(y,Di)] 'genotype'
  - Fitness/objective function mapping f(x) of each x in Di 'phenotype'
  - Selection Di →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 →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) →Di
      · Fitness calculation f(x) for each x in Di
      · Selection(di) →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 ¡- New-Priority-Queue()
  Insert (initial, q, Heuristic(initial))
  while generation limit not reached
    do current ¡- 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

The stochastic design implementation of the genetic algorithm is an effective solution in solving the drastic landscape of this complex problem, As such, there are benefits to implementing this in a steady state or generational genetic algorithm. The steady state genetic algorithm, only using two parents from the population would prioritize an increase in intensity with each generation, but drastically limit diversity at each iteration. Conversely, a generational genetic algorithm will ensure more diversity in each population but may take longer to reach a desired intensity. However, the steady state genetic algorithm may lead to epistasis and the generational genetic algorithm can explore a greater portion of the solution space.

The stochastic algorithm is more likely to quickly find a good solution to the complex problem, assuming the heuristic does not overestimate but still encompasses all measures of optimality. The downside of the stochastic algorithm is that there is the potential for the algorithm to skip over a portion of the solution space that may contain the global optimum. The computational complexity of the problem space will require less time and memory than a potential deterministic implementation. Depending on the objective, a speedy, potentially suboptimal solution, or a longer, optimal solution, one implementation may be preferred over the other.

References

[1] 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
[5] https://developers.google.com/optimization/routing/vrp
[6] 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

[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
[12] Pearl-II.pdf
[13] GA-global-search-structure.pdf
[14] Lecture2-GenSearch20.doc
[15] AStar-Example.doc
[16] Lecture4-gs-bfs-20.doc
[17] https://www.researchgate.net/post/Whats_the_difference_between_the_steady_state_genetic_algorithm_and_the_generational_genetic_algorithm
[18] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.46.827&rep=rep1&type=pdf