Project Proposal – CSCE686

Chad Willis

Updated June 20th

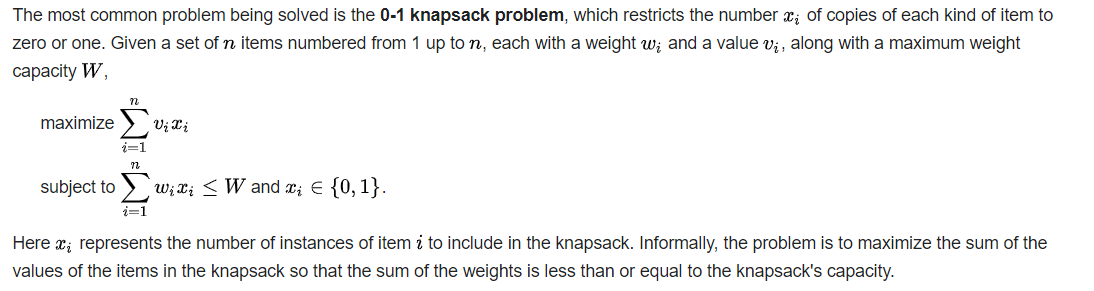
Problem Selection / Problem Domain Details

The real world problem combining two NP-Complete problems involves a single delivery van bringing groceries to local grocery stores and then going back to the starting point. The delivery driver wants to know the minimum miles needed to travel to go to every store and back. To determine this minimum amount of miles needed to travel, the traveling salesman problem can be used. At the same time, the other NP-Complete problem will be the knapsack problem. The van can only fit so many groceries into the van at max capacity. All the foods have a different weights and values. The delivery driver wants to find the maximum value of the groceries that can initially be put into the van such that the weights of the items put into the van are smaller than or equal to the max capacity.  
  
The traveling salesman problem can be defined as:  
Given a list of nodes and the weights between each pair of nodes, what is the least weight possible route that visits each node and returns to the origin node. This is different than the  [Hamiltonian Cycle](https://www.geeksforgeeks.org/backtracking-set-7-hamiltonian-cycle/). The Hamiltonian cycle problem is to find if there exist a tour that visits every city exactly once. Here we know that Hamiltonian Tour exists (because the graph is complete) and in fact many such tours exist, the problem is to find a minimum weight Hamiltonian Cycle.

One implementation of the TSP problem is as follows:

1. Consider city 1 as the starting and ending point. Since route is cyclic, we can consider any point as starting point.
2. Generate all (n-1)! permutations of cities.
3. Calculate cost of every permutation and keep track of minimum cost permutation.
4. Return the permutation with minimum cost.

The knsapsack problem can be defined as:  
Given weights and values of n items, put the items into a knapsack of capacity W to get the max total value in knapsack. Basically, you want to find the maximum value such that the weights of the items put into the bag are smaller than or equal to W. An item cannot be broken into smaller pieces; this is called the 0-1 property.   
In the case of , the Knapsack is analogous to the amount of supplies that the team can handle. The n items are all of individual supplies counted. The goal is to have the highest value, which has to do with how effective or needed the supplies are, while not going over the maximum weight allowed for the team to carry.



References

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

<https://en.wikipedia.org/wiki/Knapsack_problem>

<https://www.geeksforgeeks.org/traveling-salesman-problem-tsp-implementation/>

<https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/>

**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 all team members *j* meaning

that all drones start and end at the depot. AND ∀*di* ∈ *D* the remaining fuel in *di* ≥

the cost from the current node to the depot. AND the sum of the sizes S of items I

packed into any *di* is less than the capacity L.

***solution function*** *:* a node is a solution (terminal) if all targets in *Mt* have a drone

mapped to them AND all drones in *Md* are back at the depot *Thome* . 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 *Di*

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

“condition on z being an optimal (satiszing) 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 *Md* from drones to their current locations and *Mt* 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 *Md* to one location for each different neighbor and updating *Mt* accordingly

feasibility (n) → a node is feasible if ∀*di* ∈ *D*, *ri* ⋂ *rj* = {*Thome*}∀*i* =/ *j* meaning that all

drones start and end at the depot. AND ∀*di* ∈ *D* the remaining fuel in *di* ≥ the cost

from the current node to the depot. AND the sum of the sizes S of items I packed into

any *di* is less than the capacity L.

solution (n) → a node is a solution (terminal) if all targets in *Mt* have a drone mapped

to them AND all drones in *Md* are back at the depot *Thome* .

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 *Thome* 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 *Thome* 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.

**algorithmdomaindesignspeciﬁcation form:**

• name: A\*( Pi , Po )

• domains: Pi is set-of-candidates, Po are the sets of solutions, Pp is set of partial solutions of “generated nodes”

**– next-state-generator**

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

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

**– feasibility** ( xj , y ) → boolean [if true union (xj , y) and put result in Pp ]

**– solution** ( y ) → boolean; z = y; delay termination and and all “optimal” solutions (if satisfying accept one/first solution)

**–** objective solution( Pp ) →“ordered set/well founded set over Pp ”

**Knapsack Integration Problem domain / Algorithm Domain**

Candidates: W capacity of supplies for n total items

Next state generator: Based on cost and value, decide what should go into the teams supplies without going over weight

Selection function: select next item so that cost /value is optimized

Feasibility function: if the item can fit into the teams supplies, it is a feasible solution   
  
Solution function: if the item can fit into the teams supplies, while maximizing value and minimizing cost, it is a possible solution  
  
Objective function:

Heuristic: cost distance to return all vehicles M.

Admissibility of heuristic: List of grocery stores V will eventually be empty with all vehicles M back at the depot.

Heuristic time complexity: O(n) linear time dependent on how many vehicles M.