Project Proposal – CSCE686

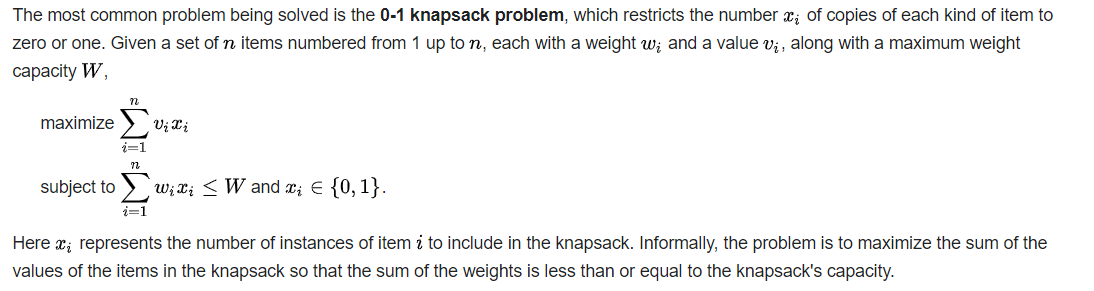
Chad Willis

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 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 van will drop off certain groceries and pick up certain groceries at the grocery stores.  
  
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.

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/>

<https://www.geeksforgeeks.org/travelling-salesman-problem-implementation-using-backtracking/>

**Phase 2:**

We have a graph G = (V,E) where are the grocery store vertices {1,…,n}. E is the edge weights for each pair of vertices, which will be the distance in miles from store to store. N will represent the amount of grocery items to be put into the van. There will be two arrays, named values and weights, which will store values and weights of the N items respectively.

Pseudocode: for knapsack:

We have a function knapsack that returns the maximum value that can be put in a knapsack of capacity W. The input to the function is W, the weights array, the values array, and N.

Base Case   
if n == 0 or W == 0

    return 0

If weight of the nth item is  more than Knapsack of capacity W,  then this item cannot be included in the optimal solution

return the maximum of two cases:

1. nth item included
2. not included

Pseudocode for TSP:

Start with a function that takes in the graph with the associated edge weights, and the source node  
Consider grocery the source node as the starting and ending point. Since route is cyclic, we can consider any point as starting point.

Generate all (n-1)! permutations of grocery stores

Calculate cost of every permutation and keep track of minimum cost permutation.

Return the permutation with minimum cost.

**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 the van, L: groceries to be put in van

- 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

***algorithmdomainrequirementsspeciﬁcationform:***

*•name: A\* (Di,Do)*

*•domains: Di isset-of-candidates,*

*Do aresetsofsolutions(solution spaceofsubsets)*

*•operations:*

*I(x); xinDi; xisapossiblecandidatefrominputset*

*O(x, z);xinDi,zinDo; zisasatisfying solution*

***algorithmdomaindesignspeciﬁcation form:***

*•name: A\* (Di,Do)*

*•domains: Di is set-of-candidates, Doarethesets ofsolutions, Dp is setofpartialsolutions*

*•imports: ADTset, list, queue,real/integer/character*

*•operations:*1

*I(x); xinDi*

*O(x, z);xinDi,zinDo;*

*“conditiononzbeingasatisfying solution”*

*I’(x, y);xinDi,yinDp; conditiononybeingapartialsolutioninDp*

*Dp isthe“open”list;Dc isthe”closed”list*

**–** *deﬁnestate*

**– *next-state-generator***

*i)* ***selection****ofapartialsolution yin DpbaseduponitssuperiorityandputinDc anddeletefromDp*

*“baseduponheuristic cost function”*

*ii)* ***Generation****ofallnextstatesxjofy*

**– *feasibility****(xj,y)−>boolean[iftrueunion (xj,y)andputresultinDp]*

**– *solution****(y) −>boolean; z=y; delaytermination andﬁnd all*

*“optimal”solutions(if satisfyingaccept first solution)*

**–** *objectivesolution (Dp) −>“orderedsetoverDp”*

**– *heuristics*** *comefromproblemdomain insight:*

***-- AttemptusePD nextstategeneratortoreduceset-ofcandidatesASAP***

***-- Attempt to generatea combination once and onlyonce incombinatorial problemdomain***

***-- Attempttogenerateearlypruningcondition simplesolutioncheck***

***algorithmdomainintermediatespeciﬁcation form: (iterative)***

*•Heuristics: distance to the next grocery store, value of groceries*

*•Data structures:input graph of grocery stores, set of edge weights between storesion), set of items weight and value, van max capacity*

*output – list of sets which is the van route and the grocery load*

***algorithmdomainfunction speciﬁcationform: (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