Project Proposal – CSCE686

Chad Willis

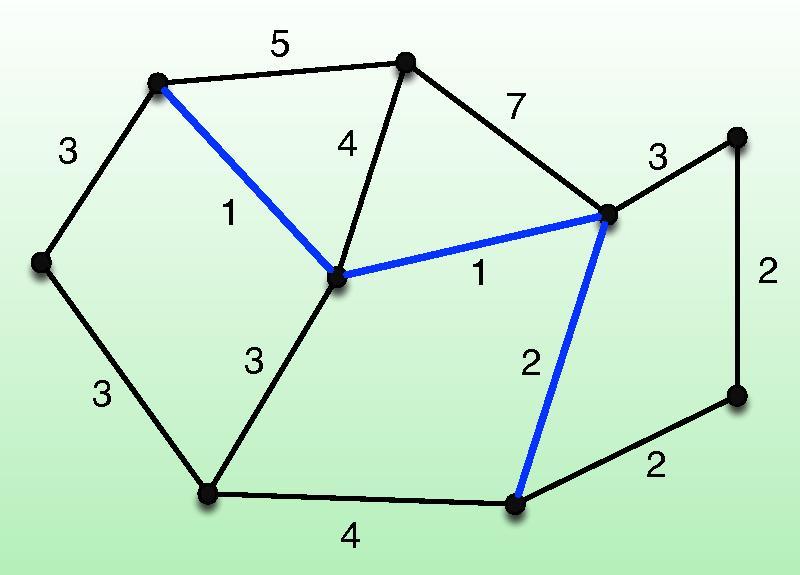
Problem Selection / Problem Domain Details

The real world problem combining two NP-Complete problems that I came up with has to do with, in a first person shooter video game, strategically holding objective points on a map. Choosing objectives will deal with the k- minimum spanning tree (k-MST) problem. K-MST will help a team of players choose a variable amount of objective points that are as close as possible together. Between the objective points will be the lowest cost edges. At the same time, the other NP-Complete problem will be the knapsack problem. The team has a total capacity of ammo, weapons, and supplies, all of which have different weights and values. The teams can only have so much capacity for the entire team. We want to find the maximum value such that the weights of the items put into the team are smaller than or equal to W.

The k- minimum spanning tree problem can be defined as:

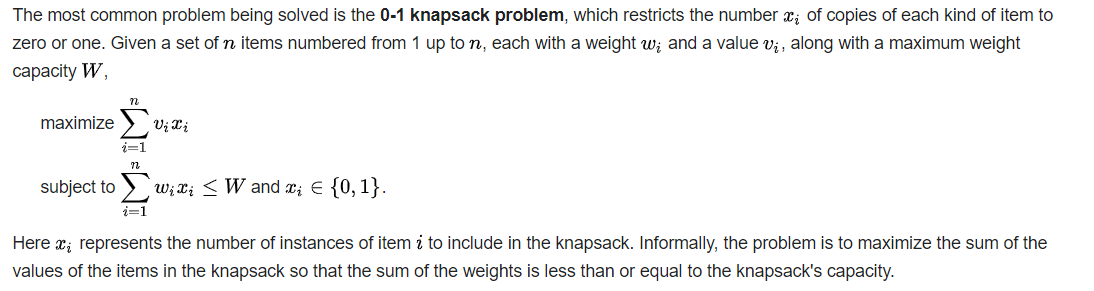
The input to the problem consists of an [undirected graph](https://en.wikipedia.org/wiki/Undirected_graph) with weights on its edges, and a number *k*. The output is a tree with *k* vertices and *k* − 1 edges, with all of the edges of the output tree belonging to the input graph. The cost of the output is the sum of the weights of its edges, and the goal is to find the tree that has minimum cost.

In the case of   
In the case choosing what objective points to take, the goal is have to lay the least expensive cables from node to node, or in this case objective to objective. How far away the objectives are determines the weight, or how expensive the edge is.  
The picture below represents the 4 MST of the graph, where 4 objectives are as close together as possible.



A basic summary of the algorithm is as follows:  
**1.** Sort all the edges in non-decreasing order of their weight.   
**2.** Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.  
**3.** Repeat step#2 until there are (V-1) edges in the spanning tree.

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/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/>

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

<https://en.wikipedia.org/wiki/K-minimum_spanning_tree>

**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.