Reid Elkins

Homework 6 - CSCE 686

June 05, 2020

Problem 2 :

**i.)** I will be using both the knapsack problem and the set covering problem with BF\* variation to pick the best subset of stocks to purchase to make a diversified portfolio. For this problem, I will consider that there is a set amount of money, only 1 share of each stock can be bought, the cost of the stock will be based on the close of market price on June 19th, 2020, the value of the stock will be based on a combination of technical factors, and each stock is part of at least one sector to be desired in the portfolio.

Given an amount of money m and a set of stocks j, each with a certain price p, value v, and sector list l, find the subset of stocks that maximizes the value while the sum of the price of the stocks is less than the money held, and the portfolio contains stocks that are in every sector.

Domains D:

Input Di: dictionary of stocks: {S: (P, V, L)}, S: stock ticker, P: stock price, V: value, L: list of sectors, M: money, G: list of all sectors desired

Output Do: list of stock tickers chosen, B

Objective function:

maximize ∑zi=1 {Si:Vi} where ∑zi=1 {Si:Vi} <= M where S is in B and z is the size of B

and

The combination of these problems is NP-Complete:

The 3-SAT problem (NP-Complete) can be reduced to the Subset Problem in polynomial time which can easily be reduced to the 0-1 Knapsack problem by setting the cost in the subset problem to the weight of each item in the knapsack problem and the given sum is now the max capacity of the knapsack. Then the set cover problem is NP, and the vertex cover problem, a known NP-complete problem, can be reduced to the set cover problem thus meaning that the set cover problem is also NP-complete. As the combination of our problems occurs in a linear type form, the combination of the two NP-complete problems is also NP-complete.

**ii.)**

***algorithm domain requirements speciﬁcation form:***

*• name: Global-Search Best First Search with a heurisitc (Di, Do); gs-bfs\**

*• domains: Di - list of stock tickers, list of prices, list of values, a list of list for sectors for each stock, integer money amount, list of all possible sectors*

*Do - max value of stocks with sum of the correlating prices less than money and the union of the set of sectors of those stocks equal to the global list of all sectors*

*• 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 an optimal (maximal) solution or set of*

*optimal solutions (or satisﬁzing solutions)*

***algorithm domain design speciﬁcation form:***

*• name: Global-Search-bfs with heuristic (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*

*• imports: ADT list, dictionary, double, integer*

*• operations:*1

*I(x); x in Di*

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

*I’(x,y); x in Di, y in Dp;*

*Dp is the “open” list;*

**–** *deﬁne state*

**– *next-state-generator***

*i)* ***selection*** *of a partial solution y in Dp based upon the highest potential total value that the solution could produce*

*ii)* ***Generation*** *of next states by adding node that includes next stock if it can be added and the node that does not include next stock*

**– *feasibility*** *(xj , y) − >* if potential cost is greater than or equal to current upper bound, and solution to Dp

**– *solution*** *(y) − > max summed value of combination of stocks where sum of the stock prices is less than or equal to the amount of money and union of sectors for chosen stocks is equal to the full list of secotrs*

**–** *objective solution(Dp) − > “ordered dictionary of nodes whose cost is greater* than or equal to the current greatest upper bound”

**– heuristics**  come from problem domain insight:

Upper bound of each node calculated as:

while <= money

Cost of each node calculated as:

where if current + > m,

Add (

Keep track of the greatest upper bound found at any node to that point.

If the found cost of any node is less than the global upper bound, kill that node

**Note:** add that value of remaining stocks are increased at every point exponentially if they belong to a sector that is not covered yet

**Then:** decrease value of stocks who belong to same sector of stock which was just added. Helps not over saturate a certain sector near the end which was left out for a while

***algorithm domain function speciﬁcation form: (iterative)***

*•* ***function*** *global-search-bfs\* (Di) sets in Do*

*• Initial condition: clear(set Dp); deﬁne initial x in Di and associated initial state s*0 *in Dp; si is search state/node.*

*Dp is the “open” list;*

*• body*

*while Di for each state not exhausted do gs-bfs\*-loop: “all nodes/states expanded”*

**– *next-state-generator***

i) find the node with the max value in the dictionary of potential nodes

*ii) “****Generation****” of all next states of s(y). Add node that includes next stock from current node as long as sum of prices of included nodes does not exceed the amount of money. Then add the node that does not include the next stock in the list to the potential node dictionary*

*delete s(y) from Dp*

**–** if **feasability** (s(xj, y)), if found cost of newly generated node is less than current upper bound, add node to dictionary of potential nodes

**–** *if* ***solution(s)*** *then save z = s, z in Do, “current optimal solution”*

**–** *end gs-dfs while loop*

**Next iteration:**

*•* ***function*** *global-search-bfs-iterative (Di) sets in Do*

*• Initial condition: clear(set Dp); deﬁne initial x in Di and associated initial state s*0 *in Dp; calculate cost and upper bound of initial x, set global upper bound to find upper bound*

*Calculation formulas:*

Upper bound of each node calculated as:

while <= money

Cost of each node calculated as:

where if current + > m,

Add (

*• body*

*while Di for each state not exhausted do gsbdfs\*-loop: “all nodes/states expanded”*

**– *next-state-generator***

i) find the node with the max cost in the dictionary of potential nodes, if multiple potential nodes have same least cost, pick the first one that was added

*delete s(y) from Dp*

*ii) “****Generation****” of all next states of s(y). Next states: one which includes the next potential stock and one which does not include the next potential stock. Generate upper bound and potential cost value of each node*

**–** if **feasability** (s(xj, y)), if found cost of newly generated node is less than current global upper bound, add node to dictionary of potential nodes

**-** if either node that is now a potential solution had a greater upper bound than the global upper bound, look through all potential nodes in current dictionary and remove those that have a cost less than the new upper bound

**–** *if* ***solution(s)*** *then it is the last node looked at before the while loop ends*

**–** *end gs-bfs\* while loop*

**Pseudocode:**

Integer money m

List stocks s of size n

List prices p of size n

List values p of size n

List of sectors groups of size n

List of all sectors

Add first stock to Di

Find the cost and upper bound of this partial solution using the formulas above

While Diis not empty

get node with lowest cost from Di (this is the best first heuristic) and set equal to

current node

**Run Knapsack part**

If there is a new upperbound:

Remove nodes from Di if their cost is greater than the new upper bound

End while

The last “current node” is the final solution

**References**

<https://www.youtube.com/watch?v=yV1d-b_NeK8>

<https://www.geeksforgeeks.org/implementation-of-0-1-knapsack-using-branch-and-bound/>

<https://www.geeksforgeeks.org/0-1-knapsack-using-branch-and-bound/>

<http://cs.franklin.edu/~shaffstj/cs319/week12.htm>

<https://www.cc.gatech.edu/~rpeng/CS3510_F16/notes/Nov28knapsackNPC.pdf>

https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/npcomplete.pdf

Lecture4\_gs\_bfs\_20.doc by Dr. Gary Lamont