Supermarket Sweep

Adam Profili, Brynn Schneberger

4/25/2021

a.

We start first with a list of the nodes given in the Supermarket Sweep.csv file, each wrapped in an Item object that holds its x, y, and price attributes.

If two items are on the same x-aisle, the shortest distance from one to the other is the absolute value of the difference between their y-coordinates. We then divide the distance by the contestant's speed, 10 feet/second to convert this calculation to time.

If two items are on different x-aisle, the shortest distance from one to the other is either by moving up to the end of the aisle, across to the other item's aisle, and then down to the item, or by moving down to the start of the aisle, across to the other item's aisle, and then up to the item. An algorithm should choose the shortest of these two options, which could be done by taking the minimum of the two or by finding if the two item's y-values on average are closer to the end (110 ft) or the beginning (0 ft). We then divide the distance by the contestant's speed, 10 feet/second to convert this distance to time.

Then two seconds is added to each time between nodes to account for the time it takes for a contestant to pick up the j^{th} node.

We implemented this in a short nested loop in Python below:

```
# initialize an empty two-dimensional list
d = [[0 for i in range(len(item_list))] for i in range(len(item_list))]
# iterate through all item objects twice for all pairings
for i in range(len(item_list)):
    for j in range(len(item_list)):
        item_i = item_list[i]
        item_j = item_list[j]

        # if item i and j share an aisle:
        if item_i.x == item_j.x:
            d[i][j] = abs(item_i.y - item_j.y) / 10
        # if they don't share an aisle:
        else:
            dist_x = abs(item_i.x - item_j.x)
            dist_y = min(((110 - item_i.y) + (110 - item_j.y) ), item_i.y + item_j.y)
            d[i][j] = (dist_x + dist_y) / 10
        # add the extra time to pick up node j
        if i!=j:
            d[j][i]+=2
            d[i][j] +=2
```

Then, since the end node we create later doesn't have an entry in the list of items, we duplicate the first row and column of the matrix to the last row and column since the end node is the same as the start node, which exists as the first item in the item list shown. If we previously added 2 to one of these entries, we want to remove it since the last node does not represent an item that needs time for collecting.

```
d.append(d[0])
for i in range(len(d[0])):
    if i in [0, len(d[0])-1]:
        d[i].append(d[i][0])
    else:
        d[i].append(d[i][0] - 2)
```

b.

Data Placeholders:

n represents the number of nodes, with node 1 being the start node, nodes 2,3,. . . ,n being item nodes, and n+1 being the end node which shares the attributes of the start node.

T represents the maximum time the contestant is given to shop.

C represents the maximum amount of items the contestant can put in their cart.

v_i represents the value of node i. $\forall i = 1, 2, ..., n$

d_{ij} represents the minimum time it takes to move from node i to node j. If node j represents an item and not a start/end point, it will include the 2 seconds to add that item to the cart. $\forall i = 1, 2, ..., n \; \forall j = 2, 3, ..., n+1$

Decision Variables:

$x_{ij} = 1$ if node j follows node i in the chosen path, 0 otherwise. $\forall i = 1, 2, ..., n \; \forall j = 2, 3, ..., n+1$

$y_j = 1$ if node j follows node i in the chosen path, 0 otherwise. $\forall j = 1, 2, ..., n+1$

$t_{ij} = y_j$ if node j follows node i in the chosen path, 0 otherwise. $\forall i = 1, 2, ..., n \; \forall j = 2, 3, ..., n+1$

$$\max_{x,y,t} \qquad \text{score} = \sum_{i=1}^{n} v_i \sum_{j=2}^{n+1} x_{ij}$$

s.t.

(1) $$y_1 = 0$$

(2) $$\sum_{j=2}^{n+1} x_{1,j} = 1$$

(3) $$\sum_{j=2}^{n+1} x_{ij} \leq 1 \qquad \forall i \in [\![2, n]\!]$$

(4) $$\sum_{i=1}^{n} x_{ij} \leq 1 \qquad \forall j \in [\![2, n]\!]$$

(5) $$\sum_{i=1}^{n} x_{i,n+1} = 1$$

(6) $$t_{ij} \leq T x_{ij} \qquad \forall i \in [\![1, n]\!] \qquad \forall j \in [\![2, n+1]\!]$$

(7) $$y_j = \sum_{i=1}^{n} t_{ij} \qquad \forall j \in [\![2, n+1]\!]$$

(8) $$\sum_{k=2}^{n+1} t_{jk} = y_j + \sum_{k=2}^{n+1} d_{jk} x_{jk} \qquad \forall j \in [\![1, n]\!]$$

(9) $$x_{ii} = 0 \qquad \forall i \in [\![1, n]\!]$$

(10) $$\sum_{i=1}^{n} x_{ij} = \sum_{k=2}^{n+1} x_{jk} \qquad \forall j \in [\![2, n]\!]$$

(11) $$\sum_{i=1}^{n} \sum_{j=2}^{n} x_{ij} \leq C$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in [\![1, n]\!] \qquad \forall j \in [\![2, n+1]\!]$$

$$t_{ij} \geq 0 \qquad \forall i \in [\![1, n]\!] \qquad \forall j \in [\![2, n+1]\!]$$

Constraint Explanations:

(1) The path begins at node 1 with a time of 0.

(2) Node 1 has exactly one destination node directly after it in the path.

(3) Nodes 2 through n may have at most one destination node directly after it in the path.

(4) Nodes 2 through n may have at most one origin node directly before it in the path.

(5) Node n+1 has exactly one origin node directly before it in the path.

(6) If the direct path from node i to node j exists, than t_ij is upper bounded at T. Otherwise, it is constrained to equal 0.

(7) Each running time y_j is set as the sum over all i of t_ij for all destination nodes j, of which at most one is nonzero.

(8) We define the sum over destination nodes k of t_jk as the sum of the running total at node j plus the additional time added by the chosen destination.

(9) No node may follow itself in the path.

(10) Nodes that are not entered may not be exited, and nodes that are entered must be exited.

(11) The amount of item nodes in the path must be less than C.

c.

Path Taken:

Node 1: Start Node (0,0)

Node 2: Coffee Beans: $6.99 at (0,15)

Node 3: K-Cups: $10.99 at (0,35)

Node 6: Granola: $5.49 at (0,100)

Node 26: Shampoo: $8.99 at (40,100)

Node 31: Trash Bags: $8.99 at (50,95)

Node 36: Air Freshner: $6.99 at (60,75)

Node 35: Dog Treats: $3.99 at (60,65)

Node 34: Broom: $13.99 at (60,35)

Node 33: Detergent: $12.99 at (60,20)

Node 40: Redbull (4): $7.99 at (70,30)

Node 39: Gatorade (12): $6.99 at (70,35)

Node 28: Paper Towels: $9.99 at (50,25)

Node 27: Toilet Paper: $7.99 at (50,15)

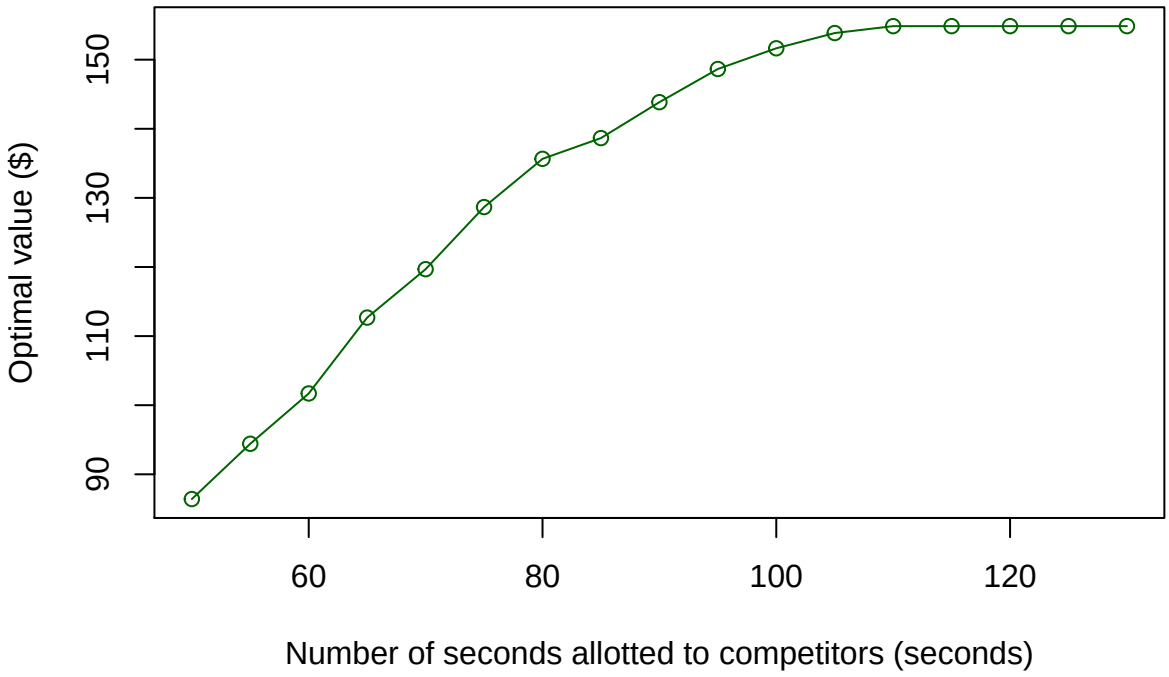
Node 22: Ibuprofen: $5.49 at (40,20)

Node 23: Diapers: $25.99 at (40,35)

Back to Start (0,0)

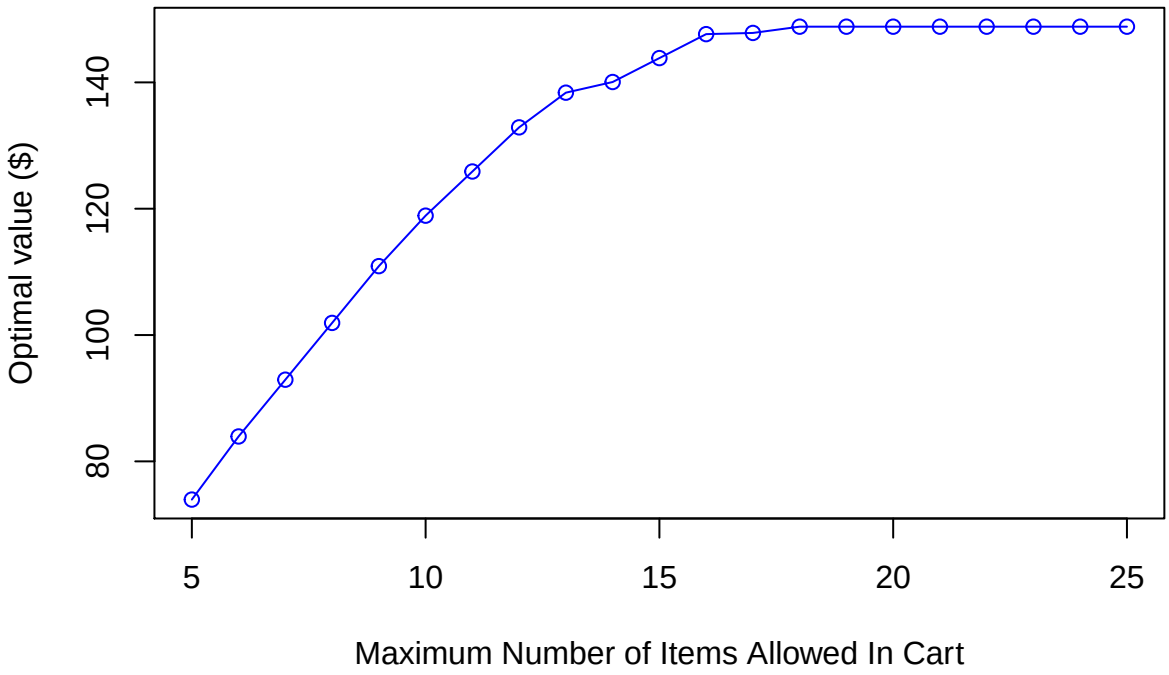
The total value of the optimized cart is $143.85

d.



e.

For this part of the problem we changed the maximum number of items allowed in the contestant's cart, while keeping the time to complete the sweep constant, at 15 seconds. As shown in the graph, the optimal value of the shopping casrt increases until the capacity of the cart is 18. At this point, the optimal value of the items in the cart plateaus and because the contestant does not have the time to go around the store and all of the expensive items. It appeaars that the most they can make during a 90 second interval is $148.82. The function of the optimal value on the number of items allowed in the cart is a concave increasing function.



f.

We changed the values of the MIPGap, a parameter that helps determine when the optimizer can terminate by giving a maximum relative percent difference in optimal values. As we changed this parameter while keeping the cart capacity and allotted time constant, we observe a decrease in the runtime, measured in seconds. We decided to preform a log base 10 trasnformation of the MIPGap to show the data better on the graph. The graph shows that as we increase the MIPGap parameter, the time to solve the probelm drasticlly decreases from a value consistently in the 45 to 50 second range all the way down to 1 second.