

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 2 seconds to the time between items if the time is not already 0. this creates an extra
        # two seconds for the contestant to pick up the item
        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])]:
        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 . $\forall i = 1, 2, \dots, n$

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. $\forall i = 1, 2, \dots, n \forall j = 2, 3, \dots, n+1$

Decision Variables:

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

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

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

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

$$\text{s.t.} \quad (1) \quad y_1 = 0$$

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

$$(3) \quad \sum_{j=2}^{n+1} x_{ij} \leq 1 \quad \forall i = 2, 3, \dots, n$$

$$(4) \quad \sum_{i=1}^n x_{ij} \leq 1 \quad \forall j = 2, 3, \dots, n$$

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

$$(6) \quad t_{ij} \leq T x_{ij} \quad \forall i = 1, 2, \dots, n \quad \forall j = 2, 3, \dots, n+1$$

$$(7) \quad y_j = \sum_{i=1}^n t_{ij} \quad \forall j = 2, 3, \dots, n+1$$

$$(8) \quad \sum_{k=2}^{n+1} t_{jk} = y_j + \sum_{k=2}^{n+1} d_{jk} x_{jk} \quad \forall j = 1, 2, \dots, n$$

$$(9) \quad x_{ii} = 0 \quad \forall i = 1, 2, \dots, n$$

$$(10) \quad \sum_{i=1}^n x_{ij} = \sum_{k=2}^{n+1} x_{jk} \quad \forall j = 2, 3, \dots, n$$

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

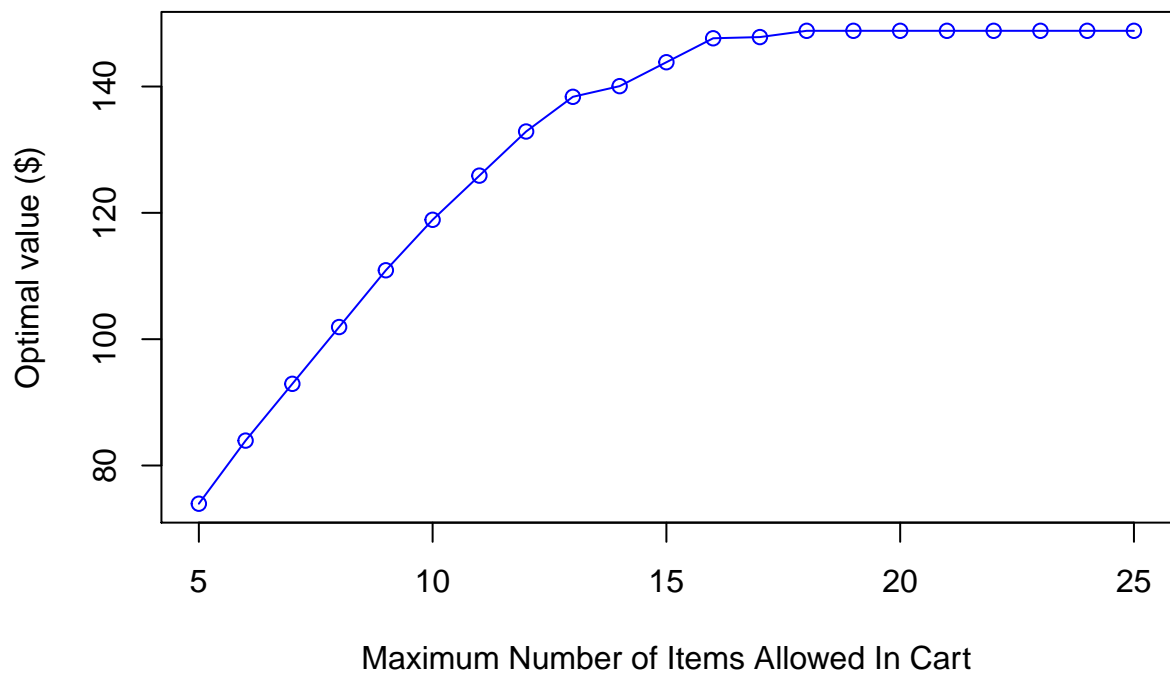
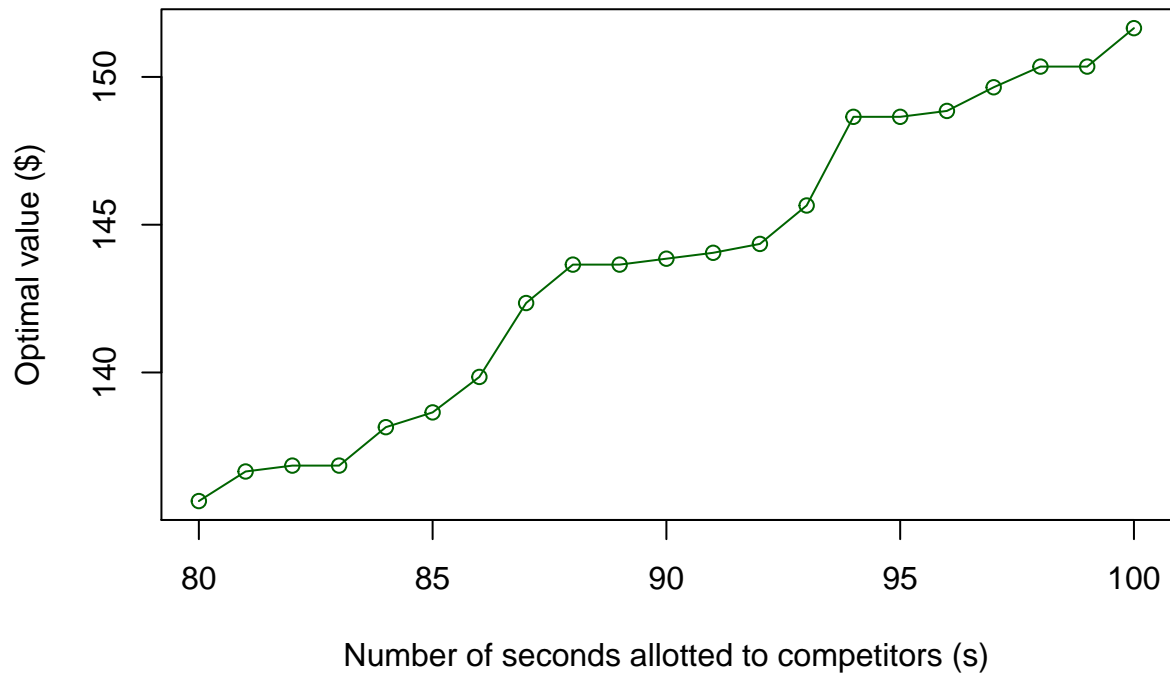
$$x_{ij} \in \{0, 1\} \quad \forall i = 1, 2, \dots, n \quad \forall j = 2, 3, \dots, n+1$$

$$t_{ij} \geq 0 \quad \forall i = 1, 2, \dots, n \quad \forall j = 2, 3, \dots, n+1$$

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, then 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 .

d.



```
f_results=c(48.907296895980835, 48.149569034576416, 50.7875018119812, 47.497539043426514, 48.5437669754  
mipgaps=c(5:15)  
plot(mipgaps, f_results, col="purple", xlab="MIP Gaps", ylab="Optimal value ($)", type='o')
```

