Route Optimization for Pick/Place Operation in Graybar's Warehouse

Adithya Jaikumar
Industrial and Enterprise Systems Engineering
University of Illinois at Urbana-Champaign
Champaign, IL 61820
jaikumr2@illinois.edu

Aishwarya Anandan
Industrial and Enterprise Systems Engineering
University of Illinois at Urbana-Champaign
Champaign, IL 61820
aananda2@illinois.edu

Haridut Athi
Industrial and Enterprise Systems Engineering
University of Illinois at Urbana-Champaign
Champaign, IL 61820
haridut2@illinois.edu

Jaydeep Chanduka
Industrial and Enterprise Systems Engineering
University of Illinois at Urbana-Champaign
Champaign, IL 61820
jkc2@illinois.edu

1 Introduction

Distribution centers and warehouses, especially the larger ones, are more often than not equipped with a Warehouse Management System (WMS) to better manage the inventory and process orders efficiently. WMS simplifies the picking (when new orders are to be satisfied) and stocking (when inventory arrives at the warehouse) processes. However due to various constraints, it might not be an entirely viable option to install WMS at every manually operated warehouse. The challenge of efficiently picking and stocking inventory in such manually operated warehouse can be very significant. Therefore, managing such warehouses and its inventory efficiently, depends on a large extent on the proper utilization of best routes to fulfill orders.

It is exactly this problem that Graybar Electric Inc is facing in its 255 manual warehouses which pretty much function on paper-pen basis at this point of time. After discussions with the the director of Graybar's Innovation Lab at the Research Park at Champaign, we understood the impact automating the routing would have in terms of non-value added activities that could be largely avoided. We were inspired by the magnitude of the problem and the scope it presented to implement a solution based on real data which could be found using the techniques discussed in this course.

Graybar Electric Inc. is North America's second largest electrical optic fiber cable distributor and a Fortune 500 company that is now 147 years old. It is predominantly a B2B type industry also providing supply chain services to its customers. It has 255 manual warehouses with a SAP framework and 18 Automated Zones installed with a fully functional Warehouse Management System. The manual warehouses range up to 90,000 sq.ft in area and walking around with a forklift to complete an order involves a lot of time and labor. Therefore, a routing algorithm that could simply compute the order in which items are to be picked or placed, would result in huge savings in terms of wasted labor and other hidden costs.

A Route Optimization problem was formulated and solved both optimally and heuristically for the biggest and busiest manual warehouse of Graybar Electric Inc at "Glendale Heights" in Illinois. The routing algorithm is built progressively starting with the simplest version of the problem and thereafter going on to add many more practical constraints.

The problem is very closely related to the classic question of "Given a set of cities and the distance between each pair of cities, what is the shortest route such that one visits each city only once and returns to the starting city? " The problem at hand closely resembles this question, except that every city is now a location of the material on the order list. Hence, this routing problem was formulated as a Traveling Salesman problem (TSP) under the assumption that at any point of time, there is only one operator to process a particular order and the capacity of the fork lift is infinite, i.e. the operator can collect all the materials in a single trip. This assumption is central to the fact that we used a Traveling Salesman Problem to figure out the optimal route instead of Vehicle Routing Problem, which demanded that we consider two or more operators processing the same order. The data used to formulate the problem, the decision variables, the mathematical formulation, the solution methodology and the results are explained in more details in the sections below.

2 Problem Definition

The lack of Warehouse Management system in a 90,000 sq.ft warehouse of Graybar at Glendale Heights, Illinois, renders the processing of system generated orders inefficient resulting in wasted labor. This work aims to transform this laborious process of pick and place operation into an efficient one by finding the shortest route an operator should take once an order is generated thereby enabling better utilization of labor.

2.1 Input data

The data used in this work is from the "Glendale Heights" warehouse of Graybar. All the data used in this problem are deterministic in nature. As with any typical TSP problem, the data used for formulation involved :

1. Warehouse layout
2. Stock Keeping Units (SKUs) in the warehouse
3. Bin location(s) for each of the SKUs
4. Randomly assumed order along with required quantity
5. Randomly assumed quantity of the material available at each location

2.2 Warehouse layout and nomenclatures involved

The layout involves a lot of bins and rack structure where the materials are located. All these racks are considered barriers through which travel is forbidden. As in every warehouse, the aisles are the only permissible path for operators to move along. The layout is shown in figure 1. Each of these bin locations are represented by an alpha-numeric sequence of 4 characters as shown in figure 2. It is important to note that even though there are racks on top of each other, we do not consider them as separate entities, thus restricting the problem to the 2-D space.

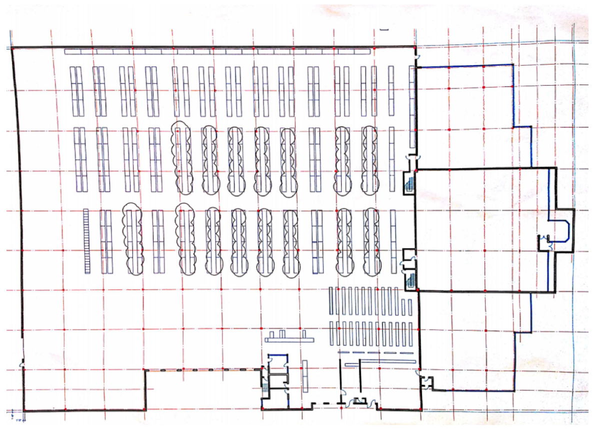


Figure 1: Layout of the warehouse used in this problem

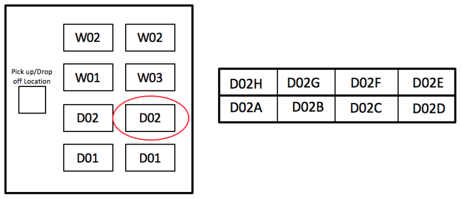


Figure 2: Example of bin location nomenclature

2.3 Distance computation

It is apparent that in such a layout, the distance metric used would be Manhattan distance otherwise known as the L1 distance metric. Though the computation of these distances between bin locations seems trivial at first sight, it is far from trivial. The procedure to calculate these distances is outlined here:

- The (x,y) coordinates of all the bin locations are listed out with respect to an origin or reference point chosen in the layout.
- A grid structure is imposed on top of the layout such that we get a graph with nodes that correspond to the grid intersections and edges correspond to the grid segments. This graph ensures that travel is forbidden through the racks.
- Once the graph is constructed, the Floyd-Warshall's all-pair shortest path algorithm is used to compute the distance matrix or the pair-wise distance between each bin location.

The same is shown in figure 3. Another thing to take note of is that pick up location of each material is at the mid-point of the outer edge of its bin location as represented by the red points in figure 6

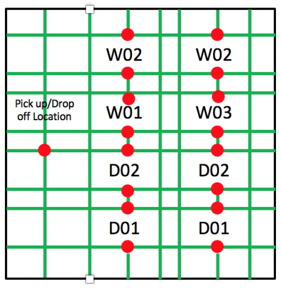


Figure 3: Imposing a grid structure on the layout to facilitate distance computation

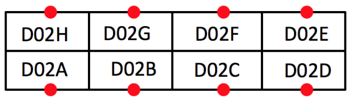


Figure 4: Pickup points at each location

2.4 Overview of the values of the input data

The data as discussed above involves the order itself, quantity required, the mapping of each material to its possible bin-location(s). There was a lot of data cleaning involved before we could work on the actual problem itself. The fact that we do not consider that multiple racks are on top of each other demanded that we transform the bin locations' nomenclature. To better understand the data cleaning involved, we provide a small subset of the data in this section to highlight the transformation of bin locations. The sample raw input data before any transformation is summarized in Table 1 and the same sample of input data after data cleaning is also highlighted in Table 2.

Material Description	Material code	Bin Location	Quantity
THHN/THWN-2 19 STR 600V 90DEG CU	22061545	W08H3	328
THHN/THWN-2 61 STR 600V 90DEG CU	22061549	W08H2	412
1 FORM 5 LB COND BODY MALLEABLE	22005033	D09I	44
FLOUR ECOLUX XL LAMP	22058641	D28D3	202
FLOUR ECOLUX XL LAMP	22058641	D28G2	463

Table 1: Sample of raw input data from Graybar

Material Description	Material code	Bin Location	Quantity
THHN/THWN-2 19 STR 600V 90DEG CU	22061545	W08H	328
THHN/THWN-2 61 STR 600V 90DEG CU	22061549	W08H	412
1 FORM 5 LB COND BODY MALLEABLE	22005033	D09I	44
FLOUR ECOLUX XL LAMP	22058641	D28D	202
FLOUR ECOLUX XL LAMP	22058641	D28G	463

Table 2: Sample of transformed input data used for our problem

The 'Quantity' column in Table 1 and Table 2 are randomly generated from a uniform distribution between 1 and 500.

The 'Bin Location' in the Table 1 take into account that at each location, there is a possibility that there can be multiple racks at different elevations each of which can store a different material. This is highlighted by the first and second entry in the Table 1 where the two different materials are stored in rack 3 (W08H3) and rack 2 (W08H2) respectively of the same bin location 'W08H'. The last digit in the bin location represents the elevation. Since, we have just considered the 2-dimensional abstraction of the problem, we transform both these bin locations into 'W08H', indicating they share the exactly the same location. The same transformation is applied to every entry in the raw data.

The other important thing to note is that there might be a possibility of the same material located at different bin locations due to various capacity restrictions of the bin. This is highlighted by the last two rows of the Table 1 where the same material 'FLOUR ECOLUX XL LAMP' is located both at 'D28D3' and 'D28G2'but in different quantities at each of these locations.

The first step in processing an order in the warehouse starts with the system generated pick ticket, contents of which is shown in the Table]ref3. It contains what materials to pick and the quantity required.

	Material description	Material code	Quantity
1	MGS400 WHITE OUTLET	22005540	10
2	ELECTRIC LINE	22007948	3
3	PULL BOX SCREW COVER	22025974	5
4	INSULATOR ONE END	22054047	10
5	FLOUR ECOLUX XL LAMP	22058641	2

Table 3: Contents of a sample order ticket

2.5 Decision variables

2.5.1 Notations

i = 1,...,M; index representing the materials in the order list
k = 1,...,K_i; index representing the bin location of material i
(i, k) represents k^{th} location of material i
N is the total number of bin locations visited in the final route
$u_{(i,k)}$ represents the relative position of the location (i, k) in the final route
$q_{(i,k)}$ represents the quantity available at location (i, k)
D_i represents the order quantity of material i

$c_{(i,k)(j,l)}$ represents the distance between location (i,k) and (j,l)

2.5.2 Definition of the variables

$$x_{(i,k)(j,l)} = \begin{cases} 1, & \text{if location } (j,l) \text{ follows location } (i,k) \text{ in the final tour} \\ 0, & \text{otherwise} \end{cases}$$

$$y_{(i,k)} = \begin{cases} 1, & \text{if location } k \text{ is selected for material } i \\ 0, & \text{otherwise} \end{cases}$$

$$u_{(i,k)} = 1, 2, ..., N$$

2.6 Problem formulation

Objective function:

$$Min : \sum_{ijkl} c_{(i,k)(j,l)} x_{(i,k)(j,l)}$$

Subject to

$$\sum_{jl} x_{(i,k)(j,l)} = 1; \forall i, k; \tag{1}$$

$$\sum_{ik} x_{(i,k)(j,l)} = 1; \forall j, l; \tag{2}$$

$$2x_{(i,k)(j,l)} \leq y_{(i,k)} + y_{(j,l)}; \forall i, j, k, l; \tag{3}$$

$$u_{(j,l)} \geq u_{(i,k)} + N x_{(i,k)(j,l)} - (N-1); \forall i, j, k, l, j \neq 1, l \neq 1; \tag{4}$$

$$\sum_{k} q_{(i,k)} y_{(i,k)} \geq D_i; \forall i; \tag{5}$$

$$u_{(i,k)} \geq u_{(i,l)}; \forall i; \text{if } q_{(i,k)} \geq q_{(i,l)} \tag{6}$$

$$u_{(1,1)} = 1; \tag{7}$$

$$x_{(i,k)(j,l)} \in \{0,1\} \forall i, j, k, l; \tag{8}$$

$$y_{(i,k)} \in \{0,1\} \forall i, k; \tag{9}$$

$$1 \leq u_{(i,k)} \leq N, u_{(i,k)} \in \mathbb{Z}, \forall i, k; \tag{10}$$

The objective is to minimize the total distance traveled for a given order list. Constraints(1) and (2) ensure that every bin location has one and only one entering and leaving leg in the final tour. Constraints (3) ensure that the edge from location (i,k) to location (j,l) is selected in the final tour only when both locations k and l are chosen for materials i and j respectively. Constraints (4) are the sub-tour elimination constraints.Constraints (5) ensure that demand for each material is satisfied. Constraints (6) are special constraints in line with the request made by Graybar to ensure that the bin locations with least quantity are emptied first if there is a choice of locations for a material. Constraint (7) makes sure that the tour starts from the order generation station (referenced as origin here on). Constraints (8), (9), (10) define the range of decision variables involved.

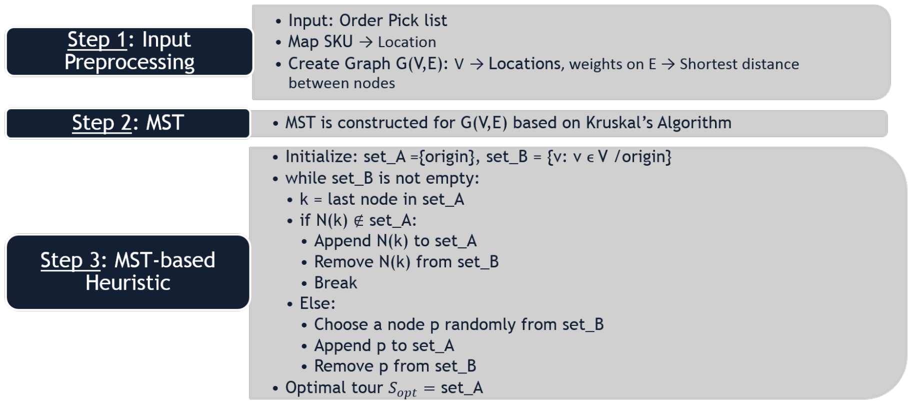
3 Solution Methodology

The Traveling Salesman Problem has been solved with many heuristics and techniques in the literature. Solving TSP optimally is feasible for only a small number of nodes. Our problem when implemented in real world would have to deal with orders that could have large quantities of materials. Hence, we use MST - based heuristic to solve the problem efficiently instead of an exact solution.The problem formulated above has many constraints that need certain modifications to any standard solution techniques in the literature. Before we try our hands at the problem formulation above, it would be beneficial to have variants of the same problem, with increasing difficulty. We broke down the above formulation into 3 variants each with its won set of assumptions. The 3 variants and their assumptions are elaborated below:

1. Variant 1: Infinite quantity at each location, single location for each material
2. Variant 2: Infinite quantity at each location, multiple locations for some materials
3. Variant 3: Finite quantity available at each location, multiple locations for some materials

3.1 Variant 1

In this variant, we consider the simplest scenario where all the materials in the order have a single location and that location has infinite quantity of that material available. We solved this problem up to optimality using Gurobi Solver for varying order quantities up to 50 materials. Beyond this point, the solver is inefficient in terms of time to solve the problem. Hence, we use MST based heuristic to reduce computation time and get an acceptable solution. The results regarding how our MST based heuristic compares with the exact solution of the Gurobi solver is discussed in the Results and Discussion section. The pseudo-code of how we solved this variant using a MST based heuristic is explained in the Figure 5.


```
Step 1: Input Preprocessing
• Input: Order Pick list
• Map SKU → Location
• Create Graph G(V,E): V → Locations, weights on E → Shortest distance between nodes

Step 2: MST
• MST is constructed for G(V,E) based on Kruskal's Algorithm

Step 3: MST-based Heuristic
• Initialize: set_A ={origin}, set_B = {v: v ∈ V /origin}
• while set_B is not empty:
    • k = last node in set_A
    • if N(k) ∉ set_A:
        • Append N(k) to set_A
        • Remove N(k) from set_B
        • Break
    • Else:
        • Choose a node p randomly from set_B
        • Append p to set_A
        • Remove p from set_B
• Optimal tour $S_{opt}$ = set_A
```

Figure 5: Pseudo-code for the solution methodology of variant 1

3.2 Variant 2

This variant of the problem is an extension of the variant 1, where we now consider the fact that some materials in the order can be present at multiple locations. It adds to the complexity because we need to find the best possible location for each of these materials to be considered for the final tour. This problem could become very hard to solve when many materials in the order have multiple locations and hence, we need to develop an approach to reduce the candidate locations for these materials. Since, this problem is still very similar to the variant 1, we adopt the MST based heuristic to solve this problem.

Intuition to eliminate candidate points

1. The warehouse layout is such that if there is a material situated in the same row but at multiple locations, both at the corners and at the middle, it would always be better to pick the material from the ones at one of the corner locations rather than the middle ones. As

can be seen from the Figure 6, an operator has to travel an additional distance to pick the material from $D02G$ or $D02F$ rather than from picking the material at $D02H$ or $D02E$ depending on the preceding location in the tour.

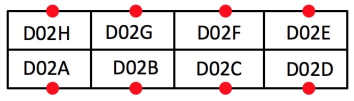
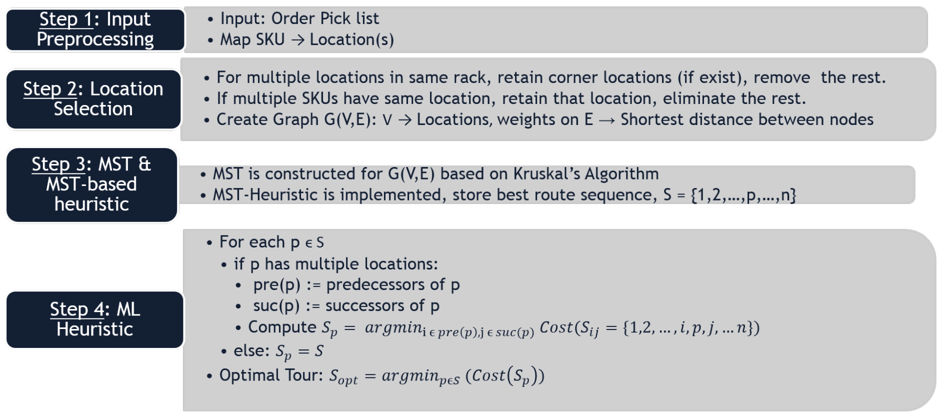


Figure 6: Example of multiple locations in the same row for the same material

2. The other intuition is that if two or more materials on the order list are located at the same location, this location has to be given more preference as more materials can be collected at the same time.
3. Finally after running the MST based heuristic (based on a random location for each material), we decided to find a better solution using a greedy approach. We refer to this heuristic as $Multiloc\ heuristic$. The greedy approach looks at the neighbors of all the materials that have multiple locations in the tour generated by the MST based heuristic after implementing the first intuition described above. We choose the best location for a particular material that has multiple locations such that the distance from its predecessor to successor(s) is minimized. This is explained by a pseudo-code shown in the Figure 7



```
Step 1: Input Preprocessing
  • Input: Order Pick list
  • Map SKU → Location(s)
Step 2: Location Selection
  • For multiple locations in same rack, retain corner locations (if exist), remove the rest.
  • If multiple SKUs have same location, retain that location, eliminate the rest.
  • Create Graph G(V,E): V → Locations, weights on E ⟶ Shortest distance between nodes
Step 3: MST & MST-based heuristic
  • MST is constructed for G(V,E) based on Kruskal's Algorithm
  • MST-Heuristic is implemented, store best route sequence, S = {1,2,...,p,...,n}
Step 4: ML Heuristic
  • For each p ϵ S
    • if p has multiple locations:
      • pre(p) := predecessors of p
      • suc(p) := successors of p
      • Compute $S_p = argmin_{i\,\epsilon\,pre(p), j\,\epsilon\,suc(p)}\ Cost(S_{ij} = \{1,2,\dots,i,p,j,\dots n\})$
    • else: $S_p = S$
  • Optimal Tour: $S_{opt} = argmin_{p \epsilon S}\ (Cost(S_p))$
```

Figure 7: Pseudo-code for the solution methodology of variant 2

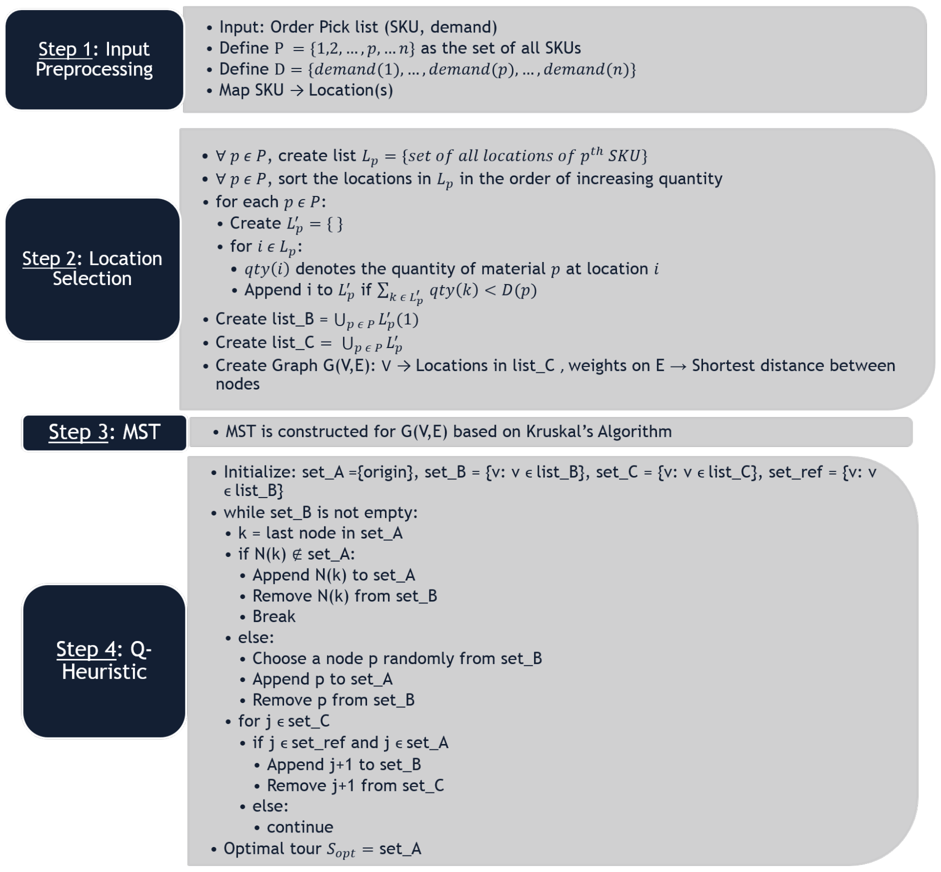
3.3 Variant 3

This variant is the one considered in the problem formulation section. This is the actual problem we intend to solve. There is a limited quantity of each material at their locations and there is also the choice of multiple locations for some materials in the order. Ideally we would like to find the shortest route but due to the requirement of Graybar, we prioritize choosing locations in the increasing order of quantity available with the goal of emptying those locations for re-stock. This is again a very complex problem and it cannot be solved optimally using standard solvers. We have developed a heuristic referred to as $Quant - heuristic$ to get a feasible and efficient solution.

3.3.1 Solution Approach

After data cleaning and identifying the order materials and quantity, all the locations of each material were sorted in the increasing order of available quantity. The locations that cumulatively satisfied the demand of each material were identified and have to be visited by the operator in the final route. So compared to the previous variants here an operator may have to visit multiple locations to collect the same material.
This is followed by creating a graph with all theses locations and finding the MST.
To create the final route shortcuts are taken in the MST. During the process of identifying the next

node in the tour, we ensure that only those nodes are considered whose priority is highest at that point of time. This is implemented in the code by introducing another set, set_C in the shortcut loop as can be seen in Figure 7.



```
Step 1: Input Preprocessing
  • Input: Order Pick list (SKU, demand)
  • Define $P = \{1,2,\dots,p,\dots n\}$ as the set of all SKUs
  • Define $D = \{demand(1),\dots,demand(p),\dots,demand(n)\}$
  • Map SKU → Location(s)

Step 2: Location Selection
  • $\forall\, p \in P$, create list $L_p = \{set\ of\ all\ locations\ of\ p^{th}\ SKU\}$
  • $\forall\, p \in P$, sort the locations in $L_p$ in the order of increasing quantity
  • for each $p \in P$:
    • Create $L'_p = \{\,\}$
    • for $i \in L_p$:
      • $qty(i)$ denotes the quantity of material $p$ at location $i$
      • Append i to $L'_p$ if $\sum_{k \in L'_p} qty(k) < D(p)$
  • Create list_B = $\cup_{p \in P} L'_p(1)$
  • Create list_C = $\cup_{p \in P} L'_p$
  • Create Graph G(V,E): V → Locations in list_C , weights on E → Shortest distance between
    nodes

Step 3: MST
  • MST is constructed for G(V,E) based on Kruskal's Algorithm

Step 4: Q-Heuristic
  • Initialize: set_A ={origin}, set_B = {v: v ϵ list_B}, set_C = {v: v ϵ list_C}, set_ref = {v: v
    ϵ list_B}
  • while set_B is not empty:
    • k = last node in set_A
    • if N(k) ∉ set_A:
      • Append N(k) to set_A
      • Remove N(k) from set_B
      • Break
    • else:
      • Choose a node p randomly from set_B
      • Append p to set_A
      • Remove p from set_B
    • for j ϵ set_C
      • if j ϵ set_ref and j ϵ set_A
        • Append j+1 to set_B
        • Remove j+1 from set_C
      • else:
        • continue
  • Optimal tour $S_{opt}$ = set_A
```

Figure 8: Pseudo-code for the solution methodology of variant 3

4 Results and Discussion

Extensive experiments were conducted for each variant type discussed above. In this section the key results are presented.

4.1 MST Based Heuristic

Experiments are conducted for randomly generated orders with quantity varying from 5 to 50. The results for the MST based heuristic are averaged over 50000 iterations for each order quantity. As mentioned before, variant 1 is implemented to optimality using Gurobi solver, but beyond an order size of 50 SKUs, the problem becomes very inefficient.The results are summarized in Table 4.

From the table 4, it can be seen that the best cost from the MST based heuristic is comparable to the optimal cost (obtained from the solver) and the computation is much more efficient. Also, the cost obtained from the MST based heuristic is compared with the Naive cost, where the Naive cost is computed as the total walking distance, when an operator (naively) picks up the SKUs according to its sequence in the randomly generated order, without any shortest route considerations. The results are also explained in the following graphs 9:

Order Quantity	Exact Approach		MST based Heuristic		% cost deviation
	Cost (ft)	Time taken (s)	Cost (ft)	Time taken (s)	
5	590.5	0.3	590.5	0.87	0
10	824	0.42	910.5	1.46	10.50
15	884	11.47	884.5	2.31	0.06
20	942	600	983	3.1	4.35
25	1069.5	455	1122	4.38	4.91
30	1254	600	1268.5	5.74	1.16
35	1386.5	600	1581.5	8.39	14.06
40	1527.5	600	1917.5	10.02	25.53
45	1641.5	214	2054.5	11.55	25.16
50	1686	315	2213.5	13.55	31.29

Table 4: Comparison of Exact Approach vs. MST based Heuristic

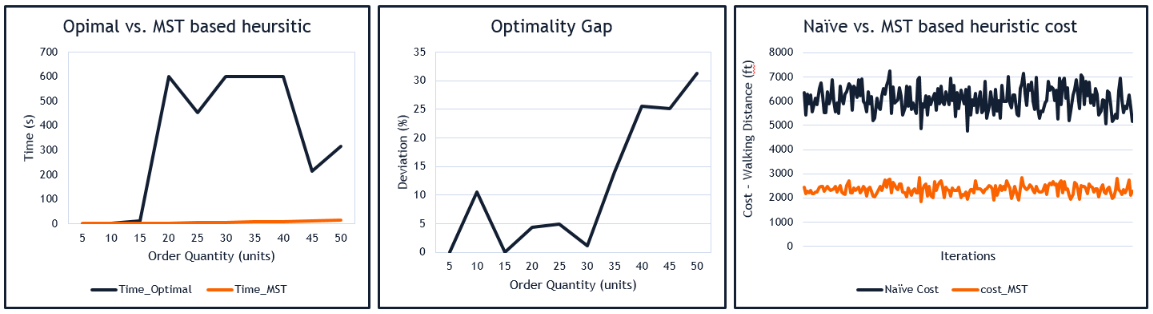


Figure 9: Variant 1 Results

4.2 Multiloc Heuristic

Experiments are conducted for a fixed order size of 50 SKUs over 200 trials and the results are plotted. From figure, it can be seen that the Multiloc heuristic performs much better compared to the Naive method, and in fact performs better than the MST based heuristic as well (for about 75% of the times).

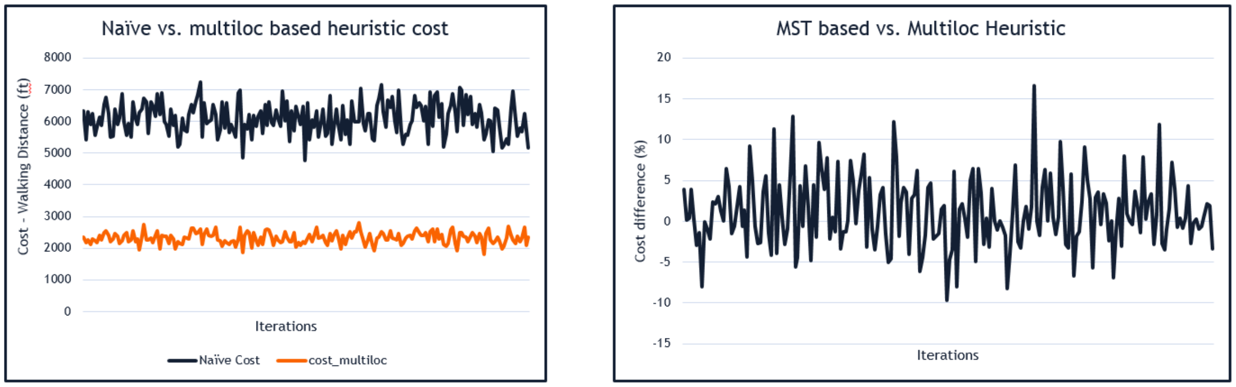


Figure 10: Multiloc heuristic results

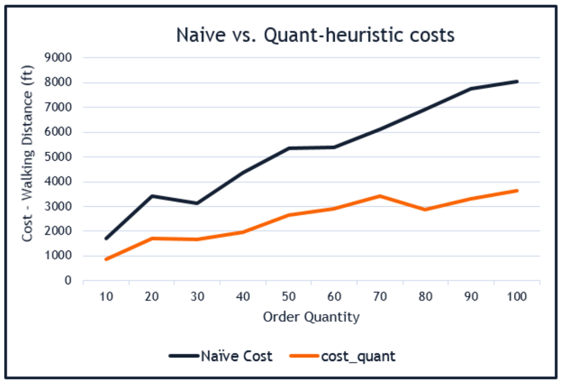
4.3 Quant-Heuristic

Experiments are conducted for randomly generated orders with quantity varying from 10 to 100. The results for the Quant-heuristic are averaged over 50000 iterations for each order quantity. The results are summarized in Table 4.

Order Quantity	Naive Cost (ft)	Quant-heuristic		% cost deviation
		Cost (ft)	Time taken (s)	
10	1708.5	871.5	2.7	48.99
20	3435.5	1723.5	4.0	49.83
30	3115.5	1679.0	7.8	46.11
40	4352.5	1962.5	12.7	54.19
50	5331.0	2642.5	11.7	50.43
60	5393.5	2903.5	17.7	46.17
70	6095.5	3434.5	18.0	43.66
80	6895.0	2877.0	16.5	58.27
90	7750.5	3295.5	24.1	57.48
100	8023.5	3648.0	28.2	54.53

Table 5: Quant-Heuristic

From the table 5, it can be seen that the cost obtained from the Quant-heuristic is much lower than the Naive cost, which is computed as before (now including the quantity). The Quant-heuristic is also computationally very efficient - even for an input size of size 100 SKUs, the computation time is of the order of a few seconds, whereas computing the optimal solution via a solver takes a lot of time (several minutes). The results are also explained in the following graphs11 .



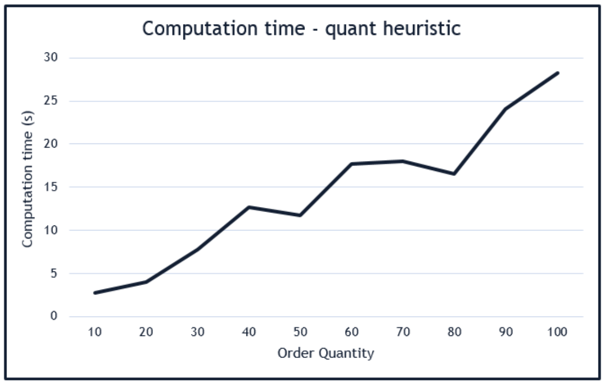


Figure 11: Quant-heuristic Results

5 Conclusion

We were able to develop a heuristic that was tailored to Graybar's requirements. The heuristic was able to provide an efficient route both in terms of the computation time and the actual cost. In this work we extend the MST based heuristic to adapt it to a warehouse setting. We believe this work would come in handy for many more similar scenarios like manufacturing shop floor routing, retail shopping, automated inventory management, etc.

6 Acknowledgements

We would like to express our gratitude to Graybar Electrical Inc. for providing us with this interesting problem and all the relevant data.

Appendix

List of variables:
i = 1,...,M; index representing the materials in the order list
k = 1,...,K_i; index representing the bin location of material i
(i, k) represents k^{th} location of material i
N is the total number of bin locations visited in the final route
$u_{(i,k)}$ represents the relative position of the location (i, k) in the final route
$q_{(i,k)}$ represents the quantity available at location (i, k)
D_i represents the order quantity of material i
$c_{(i,k)(j,l)}$ represents the distance between location (i, k) and (j, l)

Python Code for Quant-heuristic:

```
import networkx as nx, pandas as pd, numpy as np, datetime,
    itertools

t1 = datetime.datetime.now() #Start Time
order = 50 # Number of items to pick (Number on order)
# Creating a 2d array of distances between each bin location
distances = pd.read_csv('output_file.csv', header = None)
distances= np.asarray(distances)

# Adding all the bin location names to the list 'iopoints'
iopoints = pd.read_excel('Distance_info_Updated.xlsx',sheet_name
    =1)
iopoints = list(iopoints['Location'])

# Each bin location name is mapped to an index (index in distances
     matrix)
dic={}
for i in range(len(iopoints)):
     dic[iopoints[i]] = i
# Rack name mapping to Material (SKUs)
skumap = pd.read_csv('Location-SKU.csv')
#Remove all duplicates of Material and Bin Location (where both
    entries are the same)
skumap_1 = skumap.drop_duplicates(subset=['Material','BIN_LOCATION
    '], keep='first')

# Final map is a dictionary of the format {'Material no': {'Bin
    Location': [Index,Quantity]}}
final_map={}
for i in set(skumap_1['Material']):
     temp_dic={}
     bins = np.array(skumap_1[skumap_1['Material']==i]['
         BIN_LOCATION'])
     for j in bins:
          temp_dic[j] = [dic[j],skumap_1[(skumap_1['Material']==i) &
               (skumap_1['BIN_LOCATION']==j)]['Quantity'].iloc[0]]
          final_map[i] = temp_dic

# We randomly pick materials from the existing materials (Ensuring
     Origin is not in Materials)
material = np.random.choice(skumap_1['Material'][1:], order-1)
material = np.insert(material, 0, 1111, axis=0) # First material
    is 'Origin' (material no:'1111')
quant_req = np.random.randint(1,order, size=order-1) #Random
    generation of quantity required -each material in order
```

```
quant_req = np.insert(quant_req, 0, 0, axis=0) # Insert Origin's
    quantity as '0' (first material)

# Create 3 lists each with bin_location name, index and quantity
    available for each material in the order:
# A material may be available at many locations and all of these
    locations, quantity and indices are added
location=[]
indices =[]
quantity=[]
for i in material:
     location.append(list(final_map[i].keys())) #
     indices.append(list(np.array(list(final_map[i].values()))
         [:,0]))
     quantity.append(list(np.array(list(final_map[i].values()))
         [:,1]))

# Sorting multiple locations based on available quantity
# The one with the lowest quantity for a specific material comes
    first
# All the 3 lists are sorted simultaneously
for i in range(order):
     if len(location[i])>1:
          quantity[i],location[i],indices[i] = (list(t) for t in zip
              (*sorted(zip(quantity[i],location[i],indices[i]))))

# Retain only the first 'n' locations that cumulatively add up to
    the demand for each material
# All the list are updates correspondingly
for i in range(order):
     if len(location[i])>1:
          met=0
          for j in range(len(location[i])):
               met+=quantity[i][j]
               if met > quant_req[i]:
                    location[i] = location[i][:j+1]
                    quantity[i] = quantity[i][:j+1]
                    indices[i] = indices[i][:j+1]
                    break

# Creates a single list of all indices from 'indices' list for
    creating the graph nodes
single_list=[]
for i in range(len(indices)):
     single_list.append([indices[i][0]])
     if len(indices[i])>1:
          for j in range(len(indices[i])-1):
               single_list.append([indices[i][j+1]])
single_list = list(set(list(itertools.chain.from_iterable(
    single_list)))) # Unpack list of lists

G = nx.Graph() #Graph instance
G.add_nodes_from(single_list) # Add nodes from single list (nodes
    are labeled with the indices itself)
#Adding edge for all nodes except to the same node (weighted by
    distances)
for i in range(len(single_list)):
     for j in range(len(single_list)):
          if i!=j:
            G.add_edge(single_list[i],single_list[j],weight=
                distances[single_list[i],single_list[j]])

# Minimum Spanning tree by Prim's algorithm
mst = nx.minimum_spanning_edges(G, data=False)
# All the edges from the MST are added to 'edge_list'
edge_list = list(mst)

# Create a Graph out of the MST
G2 = nx.Graph()

#Creaing edges as per the MST
G2.add_edges_from(edge_list)


# MST based heuristic for solving the Traveling Salesman Problem'
'''Set_A: Intially empty, nodes get added when they are visted
   Set_B: It has all the first choice of location indices for each
        material
   Set_C: It has both the first choice and other locations for
       materials that are available at multi-locations'''
list_tour=[]
list_cost =[]
for iteration in range(20): #Repeat the process many times since
    there is certain randomness in the algorithm
    set_A = [455] # The first bin to visit will always be origin
        (455 index)
    set_B = list(set([i[0] for i in indices])) # Set_B created
        using the first index of each material from 'indices' list
    set_C = [i  for i in indices if len(i)>1] # Set_C created only
         for materials with multi-locations
    set_C = (list(itertools.chain.from_iterable(set_C))) #Unpack
    # The block removes from set_C all duplicates of indices by
        retaining only the last occurence
    # This is to maintain the order of pick-up (smallest to
        largest quantity) #Emptying bins is the priority
    set_C2 =[]
    for i in range(1,len(set_C)+1):
        if set_C[-i] not in set_C2:
            set_C2.append(set_C[-i])
    set_C = set_C2[::-1]

    #The algorithm keeps looping through until set_B is empty(All
        locations visited)
    while set_B:
        last=set_A[-1]
        flag=True
        neighbors = G2.neighbors(last) #Search for the neighbors
            of the last location of set_A in our graph
        for i in neighbors: #Search all neigbors
            if (i not in set_A) and (i in set_B): # If the
                neighbor is not in tour then
                 flag=False
                 set_A.append(i) #Add it to tour (set_A - visited)
                 set_B.remove(i) #Remove from the all-nodes (set_B)
                      list
                 break # Exit the search upon finding 'the first'
                     possible neighbor
        # Only if none of the neighbors are not candiates to enter
            , we randomly pick a location to be visited from
        # candidate list
        if flag:
            ran = (np.random.choice(set_B)) #If no neighbor is
                found, randomly jump to one node in all-node list
            set_A.append(ran) #Put the randomly picked choice to
                visted list
            set_B.remove(ran) #Remove it from the candidate list

        # All the restriction in terms of multi-location pick up
            for an item is taken care here
        # Objective is to empty the bin if possible
        for k in set_C:
            if (k in set_A) and (set_C.index(k)!=len(set_C)-1):
                if (set_C[set_C.index(k)+1] not in set_A) and (
                    set_C[set_C.index(k)+1] not in set_B):
                    set_B.append(set_C[set_C.index(k)+1])
                if (set_C[set_C.index(k)+1] in set_A):
                    set_C.remove(set_C[set_C.index(k)+1])
                set_C.remove(k)
    tour = set_A
    tour.append(455) # Add origin to the end so that tour is
        complete
    total = 0
    #Calculate distance of the tour for each iteration
    for i in range(len(tour)-1):
            total=total+distances[tour[i],tour[i+1]]
    list_tour.append(tour) # a list of all tours across iterations
    list_cost.append(total) # Cost of all tours across iterations
best_tour =list_tour[np.argmin(list_cost)] #Choose the least cost
    tour
best_cost = min(list_cost) #Least cost

# Calculate how a person would satisfy the order without any sort
    of routing (Pick material one by one as per the order)
bad_cost = 0
for i in range(len(single_list)-1):
    bad_cost += distances[single_list[i],single_list[i+1]]
bad_cost+= (distances[455, single_list[0]] + distances[455,
    single_list[-1]])
#Printing out the final route, Distance of that tour and Distance
    of walking without routing
final_route=[]
for t in best_tour:
    final_route.append([i for i in list(dic.keys()) if dic[i]==t
        ][0])
print('The shortest route:')
for i in range(len(final_route)):
    if i!=len(final_route)-1:
        print(final_route[i]+ ' -----> ', end='')
    else:
        print(final_route[i])
t2 = datetime.datetime.now()
print('Time: '+str(t2-t1))
print('Best cost: '+str(best_cost))
```