IE 312

FACILITIES DESIGN & PLANNING

Project Report

Part - I

Name & Surname	Student ID	Contribution	Signature
Saime Nur BENLİLER	2019402045	100%	
Mustafa Kutay ALMAK	2019402144	100%	
Melis TUNCER	2019402156	100%	

First look and first analysis of the project made by all 3 together. From-to and distance matrices are calculated, a constructive algorithm is applied to the problem right after the first analysis and therefore, an initial excel file is prepared. Tasks are distributed afterward. Kutay and Melis focused on the constructive algorithm and Saime focused on the improvement algorithm. Kutay and Melis improved their algorithm and applied these changes repeatedly while Saime wrote the code for the improvement algorithm and upgraded it due to the changes in the constructive algorithm and compared it with the constructive algorithm. After results are achieved the report is prepared together.

1) Introduction

The aim of this project is to design a hypothetical flexible manufacturing system (FMS) that contains 6 machines, a receiving station, a shipping station, and a central buffer area. In the FMS, these machines and the central buffer area will be placed in the locations given in Figure A.1, Appendix A.

The parts coming from receiving station will be carried by automated guided vehicles (AGVs) to the machines to be processed. After the process is done, AGVs pick them up and deliver them to the next machine or to the shipping station depending on the process plan *(Table A.1, Appendix A).* AGVs can carry one part at a time and will be moving on the unidirectional lanes between locations by using the shortest path possible. Each location has a delivery (D) and a pick-up station (P) where AGVs pick up and deliver the parts. These parts are located in the middle of lanes as can be seen in Figure A.1, Appendix A. If there is an ongoing process in a machine and a new part is on its way, it is not possible for the AGV to deliver the part assuming there is no queue capacity. So, the central buffer area is used to prevent such a deadlock and keep the parts until the machine is available again.

The layout of this facility should be determined by considering the flows and distances between the machines assuming that the operating cost is positively correlated with these variables. In other words, if the number of units moved and/or the total distance traveled between two machines increases, the operating cost of the facility also increases. Therefore, a from-to matrix and a distance matrix should be constructed initially.

Given the required operations of the parts, their arrival rate, and allocations of the operations, the from-to matrix is calculated easily. Please visit Section B.1 in Appendix B for further information regarding the calculation process and the resulting from-to matrix.

For the distance matrix, the paths from the receiving station to the delivery station of the locations are taken into account. The resulting distance matrix can be found in Section B.2 in Appendix B.

2) Analysis and Solution

Although one can comment on the required closeness of the machines from the from-to matrix, the location of the Central Buffer area is unclear since the flow to CB is ambiguous. In order to estimate the number of units moving to CB, the utilization of each route is calculated firstly:

	CB – 0	VTC1 - 1	VTC2 - 2	HMC - 3	UMC - 4	VMC - 5	SHP - 6	Shipping
Receiving		0,61	0,79					
CB - 0								
VTC1 - 1				0,42		0,17	0,17	
VTC2 - 2							0,33	0,17
HMC - 3								0,83
UMC - 4			0,17					
VMC - 5				0,20				0,70
SHP - 6		0,17		0,21	0,67	0,70		
		0,77	**0,96**	**0,83**	**0,67**	**0,87**	**0,49**	

Table 1. The utilization of each route and the machines

These utilizations can be considered as the occupancy of that route per hour. For example, the route from Receiving to VTC1 will be busy 61% of the time. On the other hand, the column sums of the table reveal the occupancy of the input queue of that machine per hour. For instance, the pickup area of VTC1 will be busy 77% of the time, of VTC2 will be occupied 96% of the time, and so on. From that information, one can also comment on the required closeness of the machines with the CB now.

In order to construct a relationship diagram between departments, a qualitative closeness rating is required. After many attempts to determine the intervals corresponding to closeness, the most effective version of the allocation is decided as such:

Occupancy Interval	Abbreviation	Closeness	Value
>0,85	**A**	Absolutely Necessary	**64**
0,70-0,84	**E**	Especially Important	**16**
0,31-0,69	**I**	Important	**4**
0-0,30	**O**	Ordinary	**1**

Table 2. Qualitative closeness rating assignments

The next step is constructing the relationship matrix with the information from the utilization matrix and closeness ratings. The basic logic is as follows:

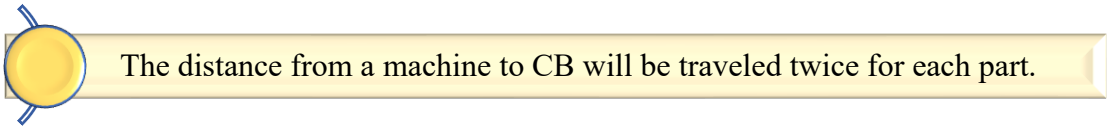
- Consider **the entries** of the utilization matrix to determine the closeness of 2 machines.
- Consider **the column sum of a machine in the utilization matrix** to determine its closeness to CB.

For example, while the utilization of 0,61 between Receiving and VTC1 corresponds to an **I** relationship between Receiving and VTC1, the total utilization of 0,96 of VTC2 corresponds to an **A** relationship between VTC2 and CB.

There is a very important point that should not be missed here. In the facility, CB is used for the part that cannot get into a machine when it arrives. When the part has waited enough in the CB, it should go back to the machine to complete its operation. This means the distance from a machine to CB will be traveled twice for each part. Therefore, while forming the relationship matrix, each relationship of CB should be written twice. For instance, the A relationship between VTC2 and CB should appear in both the entry from VTC2 to CB and the entry from CB to VTC2. With this in mind, below relationship matrix and accordingly, a relationship diagram is obtained:

	CB - 0	VTC1 - 1	VTC2 - 2	HMC - 3	UMC - 4	VMC - 5	SHP - 6	S
R		I	E					
CB - 0		E	A	E	I	A	I	
VTC1 - 1	E			I		O	O	
VTC2 - 2	A						I	O
HMC - 3	E							E
UMC - 4	I		O					
VMC - 5	A			O				E
SHP - 6	I	O		O	I	E		

Table 3. Relationships between departments



(From that point on, a number is used to refer to a machine, the letter R to refer to Receiving, and the letter S to refer to Shipping. The numbers corresponding to machines are recorded in the column and row labels of Table 3 above.)

A	(2, 0)	(5, 0)	(0, 2)	(0, 5)				
E	(R, 2)	(1, 0)	(3, 0)	(3, S)	(5, S)	(6, 5)	(0, 1)	(0, 3)
I	(R, 1)	(1, 3)	(2, 6)	(4, 0)	(6, 0)	(6, 4)	(0, 4)	(0, 6)
O	(1, 5)	(1, 6)	(2, S)	(4, 2)	(5, 3)	(6, 1)	(6, 3)	

Table 4. REL Diagram of relationships

The last step before starting implementing the constructive algorithm is calculating the Total Closeness Rating (TCR) as the sum of the values for the relationships for each department:

Dept.	A	E	I	O	TCR
0	4	4	4	0	**336**
1	0	2	2	3	43
2	2	1	1	2	150
3	0	3	1	2	54
4	0	0	3	1	13
5	2	2	0	2	162
6	0	1	4	3	35
R	0	1	1	0	20
S	0	2	0	1	33

Table 5. Total Closeness Ratings of the departments

From now on, the algorithm is executed iteratively. At the beginning of each iteration, the department to be placed is decided considering its relationships with the already placed departments. If ties exist, the department with the greatest TCR should be selected. After deciding on the department to be placed, the candidate locations for it are determined. Then, a weighted placement value (wpv) for each candidate location is calculated, and the location with the greatest wpv is selected at the end of the iteration. The process is repeated until all departments are placed.

2.1) Alternative Layout 1 with Constructive Algorithm

Since the locations of Receiving and Shipping are fixed in the beginning, the initial layout of the first iteration should be as in right:

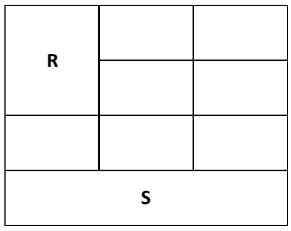


Figure 1. The initial layout of the first iteration of constructive algorithm

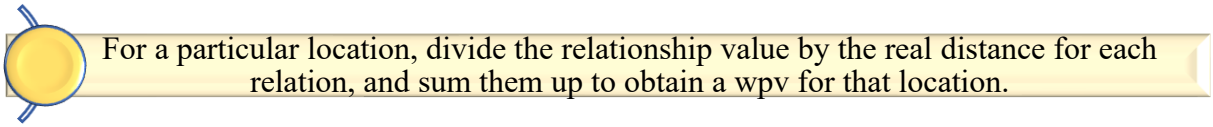
According to the constructive algorithm, the first department to be placed is chosen as the CB which has the greatest TCR among others by a large margin *(Table 5).* Then, its relationships with already placed departments are checked. As can be seen from Table 5, CB has no relationship with R or S. Therefore, it is intuitively decided that CB should be placed in the location which has the lowest distance from & to other locations. From the distance matrix *(Table B.2, Appendix B)*, the sum of the column (total distance to that location) and the sum of the row (total distance from that location) of each location is calculated. Then, location 3 is found to be the location that has the smallest sum of column and row sums.

CB should be placed to the location having the lowest distance to others since CB has no relationships with R and S.

Next, the departments having A relationship with R, S, and CB are checked *(Table 4)*. Because machine 2 and machine 5 have an A relationship with CB, they are new candidates to be placed. After checking TCR values of 2 and 5, it is decided that machine 5 with a greater TCR should be placed in the next iteration.

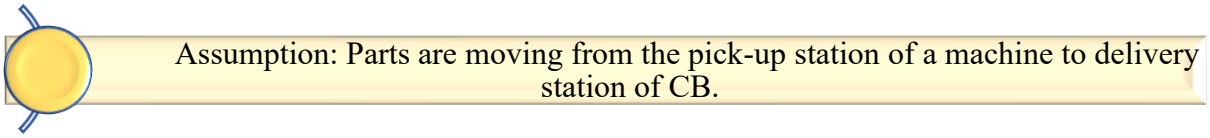
Now, it is required to calculate a wpv value for the remaining empty locations. It is beneficial to remark here that, the calculation procedure that is shown in the lecture would not yield an efficient result for this problem. The mentioned procedure assumes that the distance between the locations is exactly as they seem and calculates wpv accordingly by multiplying the value of the relationship with 1 for fully adjacent locations, and with 0.5 for partially adjacent locations. However, in this problem, the distance between 2 locations that appear to be fully adjacent to each other may be greater than the distance between another 2 locations that appear to be far from each other. Therefore, the calculation procedure for wpv's has been changed to consider the real distances. The logic is to divide the relationship value by the real distance obtained from the distance matrix for each relationship and sum them up to obtain a wpv value

for that candidate location. After all wpv's are calculated, the location having the greatest wpv is chosen. By this way, the location with stronger relationships and also lower distances would be selected.



It would be valuable to remember that the relationships are written in order by purpose in Table 4. In other words, the A relationship of (5, 0) is written for the flow from machine 5 to machine 0 and is not the same as (0, 5) which indeed refers to the flow from machine 0 to machine 5. Hence the distance from machine 5 to machine 0 is used in the calculation of wpv for the former while the reverse distance is used in the calculation for the latter. This consideration has particular importance since the distance matrix is not symmetric.

An assumption, however, arises here. The distance matrix is constructed by considering the distance from pick-up stations to delivery stations. Nonetheless, when a part stops in the input queue of a machine, it will be carried from the delivery station of that machine, where the part was last left off, to the delivery station of CB. Therefore, for example, one should divide 64 by the distance between the delivery stations of machine 5 and machine 0 when calculating the wpv component for A relationship of (5, 0). To be able to use the distance matrix in wpv calculations directly, it is assumed that the parts are traveling from the pick-up station of a machine to the delivery station of CB when calculating the relationships with CB. With this assumption, calculations can be made without further complexity.



After all calculations, the constructive algorithm is completed in 7 iterations. These intermediate iterations and their wpv calculations can be found in Appendix *(Section B.3.3).* The final layout of the constructive algorithm is generated as:

Alternative 1	
Location	M/C
LOC1	**SHP**
LOC2	**UMC**
LOC3	**CB**
LOC4	**VMC**
LOC5	**VTC1**
LOC6	**VTC2**
LOC7	**HMC**

Table 6. The final layout of the constructive algorithm

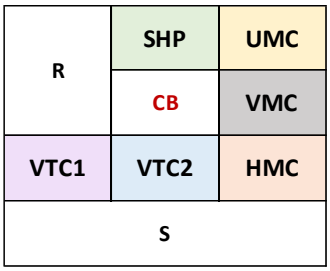


Figure 2. The final layout of constructive algorithm

For the cost calculation of layout 1, one needs to multiply the from-to flow matrix with the distance matrix entry by entry and sum the results up. However, the flow from and to CB has not been determined in the from-to matrix. Without this information, the cost calculation would be incomplete. Therefore, one last assumption regarding the flow of CB must be made.

In order to do that, the average unit flowing through relationships are calculated. For example, the number of units flowing in the routes having E relationships is summed, and the result is divided by the number of E relationships which is 8. The result which is calculated as 8,5 means that an average of 8,5 parts are moving on the routes where there's an E relationship. The methodology is applicable for I and O relationships but not for the A relationships since they appear only on the routes where either the departure or destination department is CB, whose flow is unknown. Therefore, after calculating the average units flowing for E, I, and O relationships, the average flow of the A relationship is determined as 10 by assumption.

Relationship	Avg Unit Flowing
A	**10**
E	**8,5**
I	**7,5**
O	**3,71**

Table 7. The average flows of relationships

Assumption: The average number of units flowing through a route having an A relationship is 10.

Now, all the necessary information is at hand for the cost calculation. One may find the detailed explanations of the cost calculation procedure in Section B.3 in Appendix B.

After multiplying the matrix in Table B.3.1 *(Section B.3.1 in Appendix B)* by the matrix in Table B.3.2 *(Section B.3.2 in Appendix B)* entry by entry, the below cost matrix is obtained:

Cost Matrix	6	4	**CB**	5	1	2	3	Shipping
Receiving	**0**	**0**	**0**	**0**	**336**	**468**	**0**	**0**
6	**0**	**30**	**180**	**72**	**150**	**0**	**48**	**0**
4	**0**	**0**	**180**	**0**	**0**	**90**	**0**	**0**
CB	**270**	**270**	**0**	**60**	**204**	**120**	**153**	**0**
5	**0**	**0**	**180**	**0**	**0**	**0**	**12**	**144**
1	**90**	**0**	**102**	**48**	**0**	**0**	**180**	**0**
2	**234**	**0**	**120**	**0**	**0**	**0**	**0**	**30**
3	**0**	**0**	**153**	**0**	**0**	**0**	**0**	**108**

TOTAL:	**4032**

Table 8. The final cost matrix of the constructive algorithm

As can be seen, the total cost of alternative layout 1 is found as **4032 units x meters** by multiplying all entries of the cost matrix.

2.2) Alternative Layout 2 with Improvement Algorithm

Improvement methods, also called exchange heuristics, start with an initial layout at first. Then, all pairs are temporarily swapped, and the cost matrix is calculated each time a new layout is obtained by swapping the locations of the machines. This general idea can be applied in two ways: Steepest Descent Algorithm and Descent Algorithm. Steepest Descent Algorithm calculates the cost resulting from every possible pair exchange in every iteration. Then, swap the pair that yields the maximum cost decrease. However, in the Descent Algorithm, the first encountered pair resulting in an "improvement" is swapped. If no local improvement is possible, which means a paired exchange does not yield a lower cost than the cost in the current layout, the pair exchange will not be conducted. In this project, the "Descent Algorithm" is used, and it is implemented via computer programming.

As the first step, the initial layout is determined as the alternative layout found with the constructive algorithm (*Figure 2).* The calculated cost of 4032 is the first minimum cost value in the algorithm. There are seven possible locations for 6 machines and 1 central buffer area. By using two nested while loops, all possible location exchanges are applied as (1,2), (1,3), (1,4), (1,5), (1,6), (1,7), (2,1), etc. respectively. The swapping function *(Appendix C, line 119)* is called at every iteration of nested while loops. Then, the cost is calculated by multiplying the entries of the new "From-to matrix" and the new "Distance Matrix" pairwise. Afterward, if the applied exchange leads to an improvement, which means the calculated cost with the candidate solution is lower than the cost of the current solution, the layout is updated. Then, the algorithm goes to the beginning of the nested loops by changing the values of the flag variables and swapping starts from the beginning. Else, if the applied exchange does not yield any improvement, then the exchange between this pair doesn't need to be applied. Therefore, the pair is back-swapped. *(Appendix, line 167)* The swapping continues where it left.

The stopping condition of this algorithm is "continue until a better solution, meaning that a lower cost cannot be obtained." Therefore, after the last improvement, the algorithm will continue until completing all the iterations left in the nested while loops. Then, it stops and prints the from-to matrix giving the minimum cost value.

The implemented algorithm can find two better layouts after the initial layout with a cost of 4032. The first better layout had the cost of 3780. The algorithm obtained 3780 by exchanging the locations of VTC2 and HMC. To the right, an intermediate better solution can be seen below.

Intermediate Layout	
Location	M/C
LOC1	**SHP**
LOC2	**UMC**
LOC3	**CB**
LOC4	**VMC**
LOC5	**VTC1**
LOC6	**HMC**
LOC7	**VTC2**

Table 9. Intermediate solution of improvement algorithm

Once the algorithm finds a new minimum value, it goes back to the beginning of the outer while loop and starts swapping from the beginning by accepting the new layout as the current layout. Afterward, the algorithm found a new minimum value with a cost of 3714 by changing the locations of VTC1 and SHP. After it had found a cost lower than the current cost, it updated the current layout and start swapping from the beginning. However, it performed 21 exchanges in total, which is the maximum number of possible exchanges, but it couldn't find a better solution. Therefore, we stop and conclude that **3714** is the best solution with the improvement algorithm. Below, the tabulated solution can be seen in *Table 10.*

Alternative 2	
Location	M/C
LOC1	**VTC1**
LOC2	**UMC**
LOC3	**CB**
LOC4	**VMC**
LOC5	**SHP**
LOC6	**HMC**
LOC7	**VTC2**

Table 10. The final layout of the improvement algorithm

<table>
<tr><td rowspan="2">R</td><td>VTC1</td><td>UMC</td></tr>
<tr><td>CB</td><td>VMC</td></tr>
<tr><td>SHP</td><td>HMC</td><td>VTC2</td></tr>
<tr><td colspan="3">S</td></tr>
</table>

Figure 3. The final layout of the improvement algorithm

3) Summary, Conclusion, and Further Remarks

As shown above, a constructive algorithm and an improvement algorithm are used to design the final layout. For improvement method, the descent algorithm is selected. The reason behind is that the descent algorithm considers all the possible layouts by calculating the total cost each time an exchange is applied starting from the given initial layout. Therefore, it is more unlikely to break the correctness of the final layout compared to the constructive algorithm, because the constructive algorithm takes the total cost into the account at the end instead of calculating it on the way. Since it is crucial to use a proper initial layout in the descent algorithm, the layout generated from the constructive algorithm is used to increase the robustness of the design. As a result, relatively high robustness is obtained for the design of improvement algorithm.

To compare the two algorithms, one may prefer to use the constructive algorithm since the improvement algorithm is more difficult to calculate. If the descent method is chosen, the improvement algorithm should check the C(n,2) many possible swappings per iteration in the worst case if the facility has n stations. Nonetheless, if it encounters a better solution before checking all possible changes, the solution is taken as the next base solution. The steepest descent algorithm has an even more complex procedure since it checks C(n,2) many possible swappings in every iteration and takes the best solution as the next base solution. The algorithm gets more complicated as the number of stations in the facility increases.

On the other hand, it is obvious that the improvement algorithm yields a better solution than the constructive algorithm. Therefore, it may be preferable to use improvement algorithms to reduce the incurred cost in the facility.

Appendix A

A.1) Provided Problem Definitions

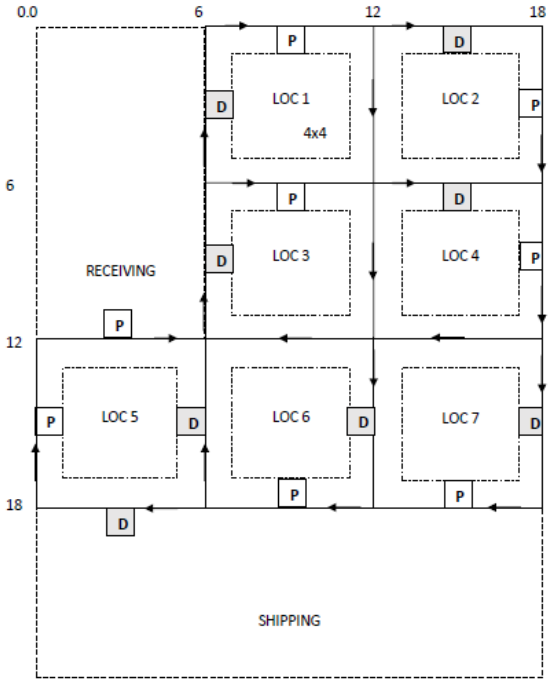


Figure A.1. Candidate machine locations together with P and D stations and the AGV flow path

Part Type	Process Plan				Arrival rate (units per hour)
A	op1	op2	op3		6
B	op4	op2	op5	op6	5
C	op7	op8	op9	op10	5
D	op11	op12	op13		2
E	op14	op8	op15		2

Table A.1. The proces plan of production

Appendix B

B.1) The From-To Matrix

The calculation procedure is as follows: part A goes into operation 1 executed by VTC2, operation 2 executed by SHP, and operation 3 executed by VMC in order. Since the arrival rate of part A is 6 units per hour, this sequence leads to the information that the required flow from Receiving to VTC2, from VTC2 to SHP, from SHP to VMC, and from VMC to Shipping will be 6. After calculating all flows, below from-to matrix is obtained:

From-to Matrix	VTC1	VTC2	CB	HMC	UMC	VMC	SHP	Shipping
Receiving	7	13						
VTC1				5		2	5	
VTC2							13	5
CB								
HMC								9
UMC		5						
VMC				2				6
SHP	5			2	5	6		

Table B.1. Required number of units per hour moving between the machines

B.2) The Distance Matrix

Distance Matrix	LOC1	LOC2	LOC3	LOC4	LOC5	LOC6	LOC7	Shipping
Receiving	12	24	6	18	48	36	30	48
LOC1	0	6	24	12	30	18	24	30
LOC2	30	0	24	36	30	18	12	30
LOC3	36	36	0	6	24	12	18	24
LOC4	24	36	18	0	24	12	6	24
LOC5	18	30	12	24	0	30	36	42
LOC6	18	30	12	24	6	0	36	6
LOC7	24	36	18	30	12	36	0	12

Table B.2. Distances between locations (in meters)

B.3) Cost Calculation

B.3.1) From-to Matrix with Flows of CB

The calculation steps are as follows:

- The rows and columns of from-to matrix (Table 1 in Appendix) is arranged according to the locations of the machines found with the constructive algorithm.
- The flow information between machines are transferred to the newly arranged from-to matrix.
- For the CB flows in from-to matrix, the averages for relationships (Table 10) are used.

Below from-to matrix is obtained after these steps:

From-to Matrix	6	4	**CB**	5	1	2	3	Shipping
Receiving					7	13		
6		5	**7,5**	6	5		2	
4			**7,5**			5		
CB	**7,5**	**7,5**		**10**	**8,5**	**10**	**8,5**	
5			**10**				2	6
1	5		**8,5**	2			5	
2	13		**10**					5
3			**8,5**					9

Table B.3.1. The final from-to matrix with correct locations and averaged CB flows

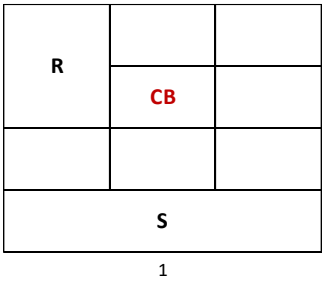
B.3.2) Adjusted Distance Matrix

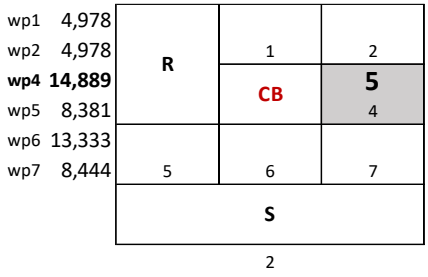
One needs to arrange the distance matrix according to the chosen locations of the machines in order to multiply the distance matrix by the from-to matrix entry by entry. Below matrix is the original distance matrix with the machine names in column & rows which location they're placed in:

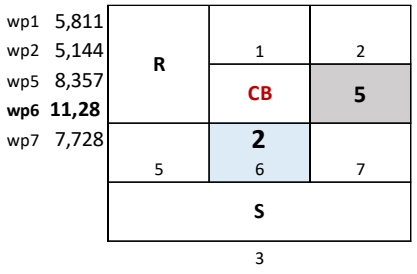
Distance Matrix	6	4	CB	5	1	2	3	Shipping
Receiving	12	24	6	18	48	36	30	48
6	0	6	24	12	30	18	24	30
4	30	0	24	36	30	18	12	30
CB	36	36	0	6	24	12	18	24
5	24	36	18	0	24	12	6	24
1	18	30	12	24	0	30	36	42
2	18	30	12	24	6	0	36	6
3	24	36	18	30	12	36	0	12

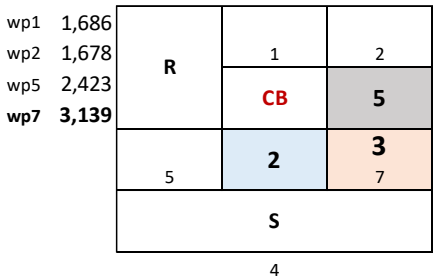
Table B.3.2. The distance matrix with machine names in locations

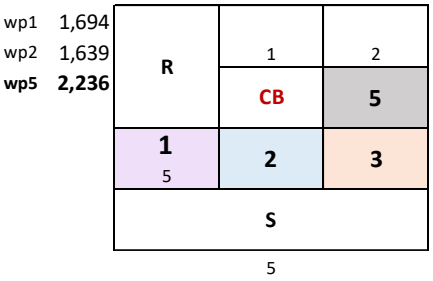
B.3.3) Intermediate Iterations and Wpv Calculations of Constructive Algorithm

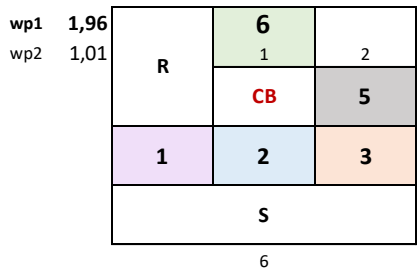












Appendix C

C.1) Code of the Improvement Algorithm

```
import numpy as np

dist_matrix = np.array(
    [
        [12, 24, 6, 18, 48, 36, 30, 48],
        [0, 6, 24, 12, 30, 18, 24, 30],
        [30, 0, 24, 36, 30, 18, 12, 30],
        [36, 36, 0, 6, 24, 12, 18, 24],
        [24, 36, 18, 0, 24, 12, 6, 24],
        [18, 30, 12, 24, 0, 30, 36, 42],
        [18, 30, 12, 24, 6, 0, 36, 6],
        [24, 36, 18, 30, 12, 36, 0, 12],
    ]
)

from_receiving = [
    ["SHP", 0],
    ["UMC", 0],
    ["CB", 0],
    ["VMC", 0],
    ["VTC1", 7],
    ["VTC2", 13],
    ["HMC", 0],
    ["Shipping", 0],
]

from_SHP = [
    ["SHP", 0],
    ["UMC", 5],
    ["CB", 7.5],
    ["VMC", 6],
    ["VTC1", 5],
    ["VTC2", 0],
    ["HMC", 2],
    ["Shipping", 0],
]

from_UMC = [
    ["SHP", 0],
    ["UMC", 0],
    ["CB", 7.5],
```

```
    ["VMC", 0],
    ["VTC1", 0],
    ["VTC2", 5],
    ["HMC", 0],
    ["Shipping", 0],
]

from_CB = [
    ["SHP", 7.5],
    ["UMC", 7.5],
    ["CB", 0],
    ["VMC", 10],
    ["VTC1", 8.5],
    ["VTC2", 10],
    ["HMC", 8.5],
    ["Shipping", 0],
]

from_VMC = [
    ["SHP", 0],
    ["UMC", 0],
    ["CB", 10],
    ["VMC", 0],
    ["VTC1", 0],
    ["VTC2", 0],
    ["HMC", 2],
    ["Shipping", 6],
]

from_VTC1 = [
    ["SHP", 5],
    ["UMC", 0],
    ["CB", 8.5],
    ["VMC", 2],
    ["VTC1", 0],
    ["VTC2", 0],
    ["HMC", 5],
    ["Shipping", 0],
]

from_VTC2 = [
    ["SHP", 13],
    ["UMC", 0],
    ["CB", 10],
    ["VMC", 0],
```

```
    ["VTC1", 0],
    ["VTC2", 0],
    ["HMC", 0],
    ["Shipping", 5],
]

from_HMC = [
    ["SHP", 0],
    ["UMC", 0],
    ["CB", 8.5],
    ["VMC", 0],
    ["VTC1", 0],
    ["VTC2", 0],
    ["HMC", 0],
    ["Shipping", 9],
]


def get_index(part):
    for k in range(8):
        if from_receiving[k][0] == part:
            return k + 1
    return None


def find_machine_location(machine):
    if machine != "Receiving":
        return get_index(machine)
    elif machine == "Receiving":
        return 0


def swap_rows(first, second):
    from_UMC[first], from_UMC[second] = from_UMC[second], from_UMC[first]
    from_CB[first], from_CB[second] = from_CB[second], from_CB[first]
    from_VTC1[first], from_VTC1[second] = from_VTC1[second], from_VTC1[first]
    from_VTC2[first], from_VTC2[second] = from_VTC2[second], from_VTC2[first]
    from_SHP[first], from_SHP[second] = from_SHP[second], from_SHP[first]
    from_HMC[first], from_HMC[second] = from_HMC[second], from_HMC[first]
    from_VMC[first], from_VMC[second] = from_VMC[second], from_VMC[first]
    from_receiving[first], from_receiving[second] = (from_receiving[second],
from_receiving[first])

#Multiplication to obtain cost matrix
def get_min():
    minValue = 0
```

```
    for j in range(8):
        minValue += (
            dist_matrix[find_machine_location("UMC")][j] * from_UMC[j][1]
            + dist_matrix[find_machine_location("CB")][j] * from_CB[j][1]
            + dist_matrix[find_machine_location("VTC1")][j] * from_VTC1[j][1]
            + dist_matrix[find_machine_location("VTC2")][j] * from_VTC2[j][1]
            + dist_matrix[find_machine_location("SHP")][j] * from_SHP[j][1]
            + dist_matrix[find_machine_location("HMC")][j] * from_HMC[j][1]
            + dist_matrix[find_machine_location("VMC")][j] * from_VMC[j][1]
            + dist_matrix[find_machine_location("Receiving")][j] *
from_receiving[j][1]
        )
    return minValue


#Swap rows until you get a new MinVal, if so, change the layout and start from
scratch
i = 0
newMin = 0
check = 0 #flag variable
minVal = get_min()
print("Initial cost: ", minVal)
while i < 6:
    k = i + 1
    while k < 7:
        swap_rows(i, k)

        newMin = get_min()
        if newMin < minVal:
            #start from scrath,a better cost is obtained.
            i = 0
            k = 1
            minVal = newMin
            print("newMin = ", minVal)
            check = 1
            break
        else:
            # if there is no improvement, back swap the locations, keep going with
your old layout.
            swap_rows(k, i)
            k += 1 #keep swapping the locations, you cannot find a better cost.

    #a better cost cannot be found, keep swapping by changing index in the outer
loop.
    if check == 0:
```

```
        i += 1
    #a better cost can be found, go to beginning with your new accepted layout.
    else:
        i = 0
        check = 0

# print all the values
print("Receiving: ", from_receiving)
print("VMC: ", from_VMC)
print("CB: ", from_CB)
print("HMC: ", from_HMC)
print("VTC1: ", from_VTC1)
print("SHP: ", from_SHP)
print("VTC2: ", from_VTC2)
print("UMC: ", from_UMC)

print("Final cost: ", minVal)
```

C.2) The Output of the Code

```
Initial cost:  4032.0
newMin =  3780.0
newMin =  3714.0
Receiving:  [['VTC1', 7], ['UMC', 0], ['CB', 0], ['VMC', 0], ['SHP', 0], ['HMC', 0], ['VTC2', 13], ['Shipping', 0]]
VMC:  [['VTC1', 0], ['UMC', 0], ['CB', 10], ['VMC', 0], ['SHP', 0], ['HMC', 2], ['VTC2', 0], ['Shipping', 6]]
CB:  [['VTC1', 8.5], ['UMC', 7.5], ['CB', 0], ['VMC', 10], ['SHP', 7.5], ['HMC', 8.5], ['VTC2', 10], ['Shipping', 0]]
HMC:  [['VTC1', 0], ['UMC', 0], ['CB', 8.5], ['VMC', 0], ['SHP', 0], ['HMC', 0], ['VTC2', 0], ['Shipping', 9]]
VTC1:  [['VTC1', 0], ['UMC', 0], ['CB', 8.5], ['VMC', 2], ['SHP', 5], ['HMC', 5], ['VTC2', 0], ['Shipping', 0]]
SHP:  [['VTC1', 5], ['UMC', 5], ['CB', 7.5], ['VMC', 6], ['SHP', 0], ['HMC', 2], ['VTC2', 0], ['Shipping', 0]]
VTC2:  [['VTC1', 0], ['UMC', 0], ['CB', 10], ['VMC', 0], ['SHP', 13], ['HMC', 0], ['VTC2', 0], ['Shipping', 5]]
UMC:  [['VTC1', 0], ['UMC', 0], ['CB', 7.5], ['VMC', 0], ['SHP', 0], ['HMC', 0], ['VTC2', 5], ['Shipping', 0]]
Final cost:  3714.0
```