B609 Project Spring 2013

Implementation of Hungarian Job Assignment Problem

Rakesh Reddy Vanga

Sucharitha Srirangaprasad

Contents

Table of figures

1. Purpose of the project

The aim of the project is to efficiently implement the Hungarian algorithm to solve the job assignment problem such that when an input of the cost matrix is given an optimal solution with minimum cost is found out. Additionally the system must ask the user for input on any special cases the user might want to implement like maximizing the profit for a given cost matrix instead of minimizing. The solution which includes the job assignment and the optimal cost must be provided to the user. Multiple solutions must be displayed if they exist.

2. Software Requirements

Programming Language used: Java

Eclipse 1.5.0

JDK 1.7

RAM 4 GB

3. Problem Description

The purpose of the assignment problem is to minimize the total cost required to perform n different tasks by m different people where each person requires some cost to perform a task. It is a special case of the transportation problem. The assignment problem is generally used in companies for planning. It can be used to solve longest/shortest route models. It can be used to assign people to jobs, assign salesmen to sales territories and also to solve the travelling salesman problem.

4. Solutions to the Assignment Problem

There are different methods in which the Hungarian Problem can be solved. They are Enumeration method, Simplex method, Transportation method and Hungarian method.

4.1 Enumeration Method

If there are n facilities and n jobs, we can have n! possible assignments. Hence to find the optimal solution, all these assignments must be written down and the assignment with the lowest cost must be considered as the optimal solution. This works well when n is small. But the size of n is large, this algorithm is inefficient.

4.2 Simplex Method

Assignment Problem can be converted to an integer linear programming problem because every assignment problem can be formulated as a 0 or a 1. Hence the simplex method can also be used to solve such a problem. Here we will have $n \times n$ decision variables and $n+n$ or $2n$ equalities. This type of solution again works for small values of n but becomes difficult as the size of n increases.

4.3 Transportation Method

Assignment is a special case of Transportation problem and hence it can also be solved using the transportation methods. Hence if there is a matrix of size n the number of assignments which will be possible are $m+n-1= n+n-1= 2n-1$ assignments. But according to the problem statement only n assignments are allowed. Hence the assignment problem is said to be inherently degenerate. In order to solve the assignment problem using transportation problem $(n-1)$ number of extra dummy allocations must be used. This is computationally inefficient and hence the transportation method is not preferred in order to solve the assignment problem.

4.4 Hungarian Method

Due to the disadvantages present in the above mentioned methods we use a special algorithm which can find the optimal solution in polynomial time and also efficiently for all the cases. This is called the Hungarian method. This method was developed by Harold Kuhn in 1955 based on the earlier works of Dénes Kőnig and Jenő Egerváry who were two Hungarian mathematicians. Hence the algorithm is called Hungarian Algorithm.

5. Assignment problem formulation

5.1 Problem Formulation

Consider that a company has n people who have different costs to perform different jobs. Let us consider that the number of jobs is equal to the number of people which is n. Each person can perform only one job.

To formulate the assignment problem in mathematical programming terms, we define the activity variables as

$$x_{ij} = \begin{matrix} 1 \text{ if job j is performed by worker i} \\ 0 \text{ otherwise} \end{matrix}$$

for i = 1, 2, ..., n and j = 1, 2, ..., n

Let c_{ij} is the cost of performing jth job by ith worker.

The optimization model is

Minimize $c_{11}x_{11} + c_{12}x_{12} + \text{-------} + c_{nn}x_{nn}$

subject to

$x_{i1} + x_{i2} + \text{..........} + x_{in} = 1$ $\quad$ i = 1, 2,......., n

$x_{1j} + x_{2j} + \text{..........} + x_{nj} = 1$ $\quad$ j = 1, 2,......., n

$x_{ij} = 0$ or 1

In S Sigma notation

minimize $c_{ij}x_{ij}$ such that

x_{ij} = 1 for i = 1, 2, ..., n

x_{ij} = 1 for j = 1, 2, ..., n

x_{ij} = 0 or 1 for all i and j

5.2 Assumptions

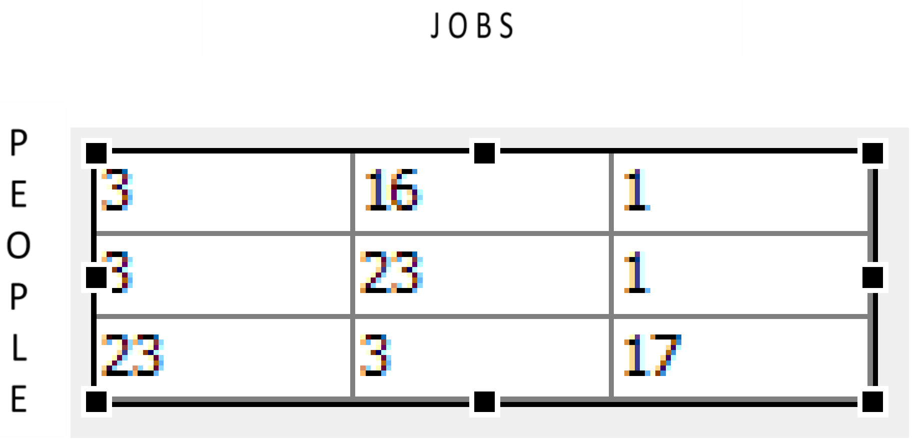
- The number of jobs and the number of machines or persons are equal.
- Each person is assigned only one job.
- Each person is independently capable of handling any job.
- Assignment criteria such as minimizing cost or maximizing profit is clearly specified.

6. Hungarian Method

6.1 Input

The input to the algorithm is a set of people and jobs and the cost each person takes to complete every job. The output of the algorithm is the job that is assigned to every person and the sum of all the costs i.e. the total cost required to complete the entire set of jobs.

Consider the example shown below. The people are represented as rows of the matrix and the jobs are represented as the columns of the matrix. The cost that person i takes in order to perform a job j is the element C_{ij} of the matrix.


JOBS
PEOPLE

3	16	1
3	23	1
23	3	17

Fig 6.1: Input entered

The input given by the user is converted to an m*n cost matrix as shown below. The algorithm is applied on this matrix.

$$\begin{array}{cc} & \text{JOBS} \\ \begin{matrix}\text{P}\\\text{E}\\\text{O}\\\text{P}\\\text{L}\\\text{E}\end{matrix} & \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n} \\ \vdots & \vdots & & \vdots \\ c_{n,1} & c_{n,2} & \cdots & c_{n,n} \end{bmatrix} \end{array}$$

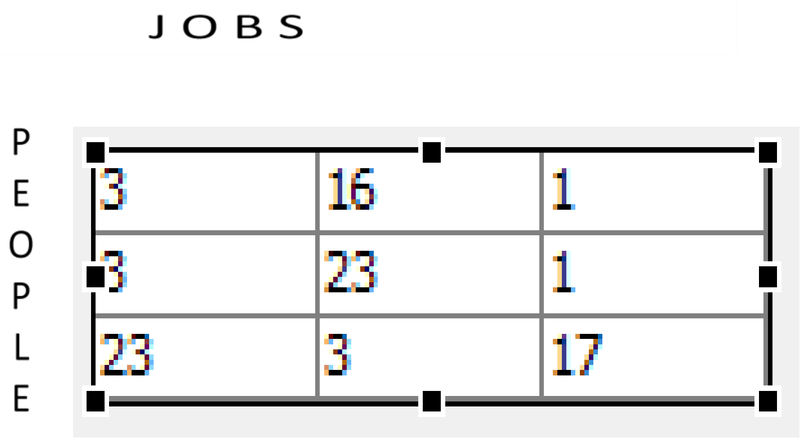
Fig 6.2: Cost Matrix Format

6.2 Principle

The theorem or the principle used in solving the problem is as given below:

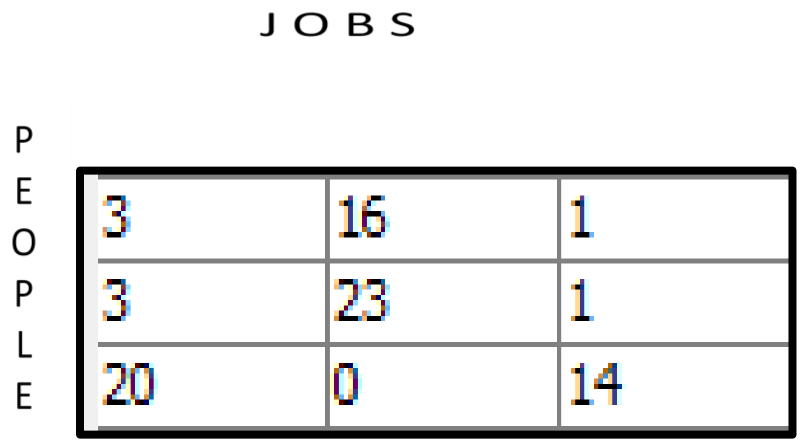
- An optimal solution is computed for a matrix, which is produced by the addition/subtraction of a number to the elements in the original matrix.
- This optimal solution will be equivalent to the solution computed on the original matrix.

  Consider the example shown below. The first matrix is the original matrix. The second matrix is obtained by subtracting the minimum element in the third row which is 3 from every element in the third row. An optimal solution obtained on the second matrix will be equal to the optimal solution obtained on the first matrix.


JOBS
PEOPLE

3	16	1
3	23	1
23	3	17

Fig 6.3: Original Cost Matrix


JOBS
PEOPLE

3	16	1
3	23	1
20	0	14

Fig 6.4: Cost Matrix after Reduction

6.3 Output

The output comprises of all the possible optimal solutions along with the total cost to perform all the jobs.

6.4 Algorithm

The algorithm of the Hungarian method is as follows:

1. The minimum element in every row is found out and subtracted from every element in that row.
2. The minimum element in every column is found out and subtracted from every element in that column.
3. The assignment is done on the matrix obtained after applying the first two steps in the following way:
    i. For every row or a column which has only one zero which has not been assigned already or eliminated, we box that zero value and consider it as assigned.
    ii. For every zero value that is assigned, we cross out all the zeroes present in the same row or column and mark them as eliminated.
    iii. If for a particular row or a column, there are two or more zeroes and if we cannot choose one of them by inspection, then we can choose one zero arbitrarily and assign it.
    iv. The above steps are executed until every zero in the matrix is either assigned or eliminated.
4. If the number of assigned zeroes is equal to the number of people or jobs then an optimal assignment is found. If there are instances where zeroes have been chosen arbitrarily, then there may be different optimal solutions. If an optimal solution is not found then we go to step 5.
5. We have to now cover all the zeroes which are obtained through step 3 by using the following steps:
    i. All the rows which are unassigned must be marked.
    ii. Mark all the columns which have zeroes in the unassigned (marked) rows and which have not already been marked.
    iii. Mark all the rows which are unmarked and which have assignments in the marked columns.
    iv. Repeat the above three steps until no rows or columns can be marked.
    v. Draw lines through all unmarked rows and marked columns.

6. Select the minimum element from all the elements through which a line has not been drawn. Subtract this minimum element from all the elements which do not have a line passing through them and add it to the elements which have two lines passing through them i.e. which have a line passing through the row as well as the column of that element. Now we have a new matrix on which assignment can be performed.
7. Start again from Step 3 and repeat all the steps until an optimal assignment is obtained.

6.5 Code

The Hungarian Algorithm implementation consists of the following functions:

1. The function initial() performs the first two steps of the Hungarian Algorithm. It performs the following operations:

- finds the minimum element in each row
- subtracts the minimum element in every row from the entire row
- finds the minimum element in each column
- subtracts the minimum element in every column from the entire column

2. The function assignment() performs the assignment after the initial reduction of the cost matrix and also after every reduction. Assignment is performed if:

- if the number of zeroes in a row is one and its columnLabel shows no assignment assign it
- if the number of zeroes in a column is one and its columnLabel and rowLabel show no assignment assign it
- if there are zeroes whose columnLabel and rowLabel show no assignment assign them

3. The function ticking() which performs ticking based on the following rules:

- Tick all unassigned rows
- In each ticked row, if there are zeroes present, tick the corresponding columns
- For each ticked column, if there are any assignments present, tick the corresponding row
- Perform the second and the third steps until no further ticking is possible

4. The function change() which changes the cost matrix according to the following conditions using the minimum element obtained after ticking

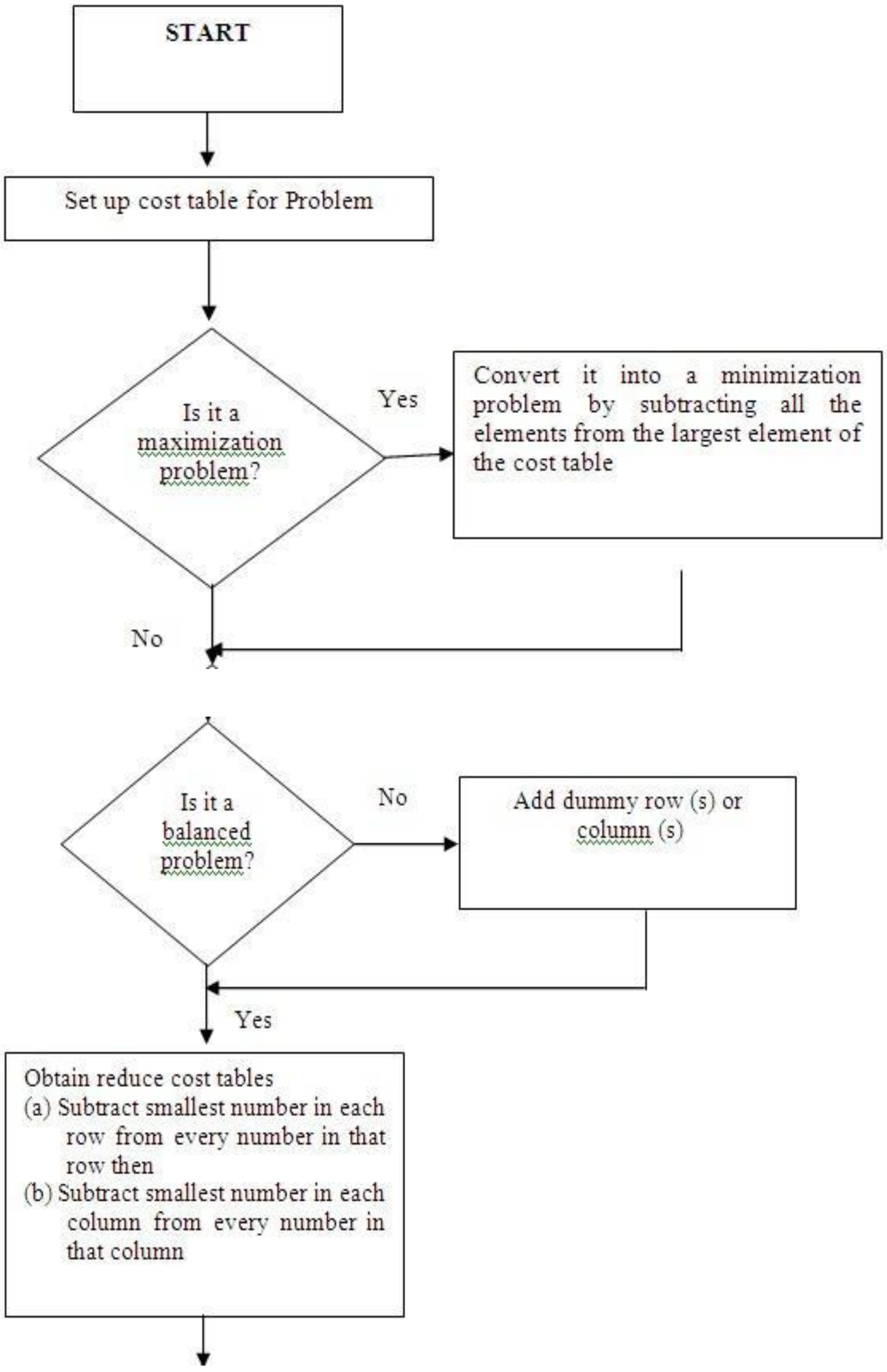
- Subtracts the minimum element from the cost matrix element if its row is ticked and column is not ticked
- Adds the minimum element to the cost matrix element if its row and column are ticked
- Leaves the element of the cost matrix unchanged if its row and column are not ticked

5. The function finalDisplay() which displays the table with the set of people and jobs assigned to each person for each optimal solution. It also displays the total cost of the optimal solution.

6. AlternateFunction() This function is used to implement multiple solutions, if any, by using the algorithm Munkres, J. Algorithms for the Assignment problem [2]. The output dialog box displays all the possible outputs for the given matrix.

All the special cases implemented in this project are refered from Harold W. Kuhn. Variants of the Hungarian method for assignment problems [3].

6.6 Workflow Diagram


START
Set up cost table for Problem
Is it a maximization problem?
Yes
Convert it into a minimization problem by subtracting all the elements from the largest element of the cost table
No
Is it a balanced problem?
No
Add dummy row (s) or column (s)
Yes
Obtain reduce cost tables
(a) Subtract smallest number in each row from every number in that row then
(b) Subtract smallest number in each column from every number in that column

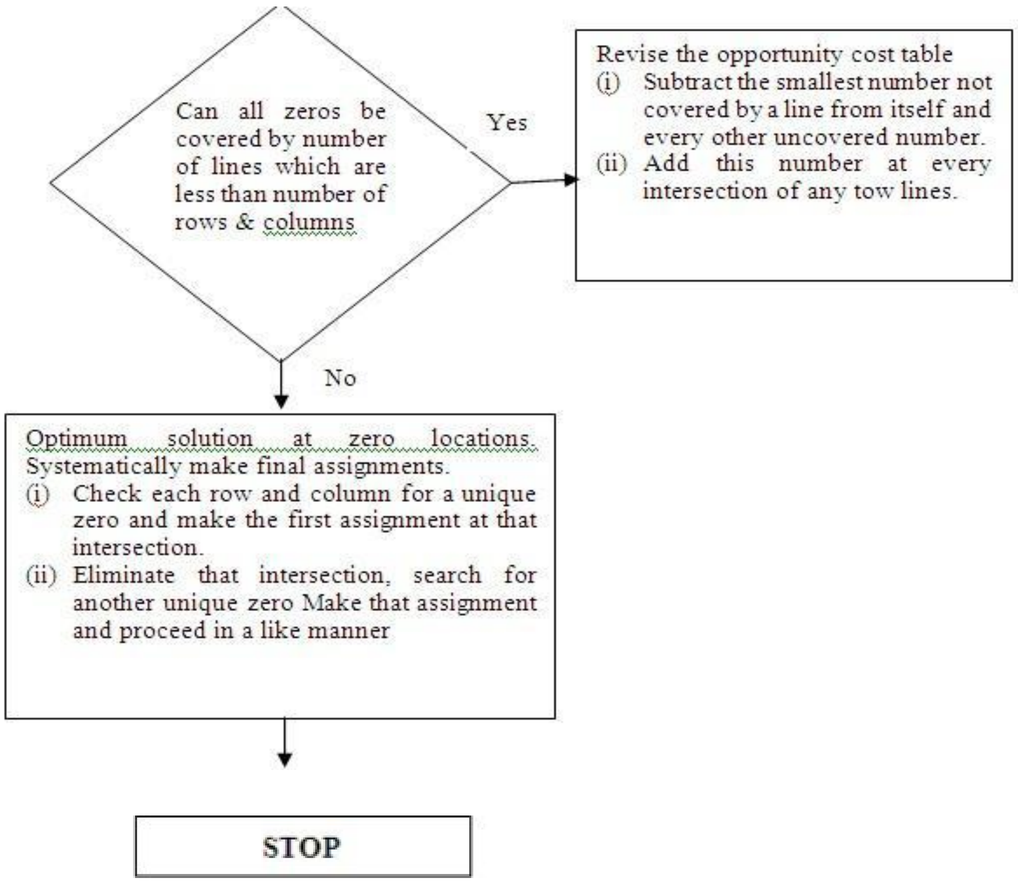
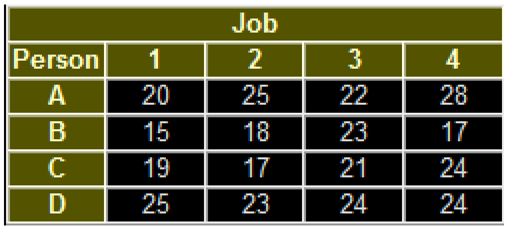

Can all zeros be covered by number of lines which are less than number of rows & columns
Yes
Revise the opportunity cost table
(i) Subtract the smallest number not covered by a line from itself and every other uncovered number.
(ii) Add this number at every intersection of any tow lines.
No
Optimum solution at zero locations. Systematically make final assignments.
(i) Check each row and column for a unique zero and make the first assignment at that intersection.
(ii) Eliminate that intersection, search for another unique zero Make that assignment and proceed in a like manner
STOP

Fig 6.5: Workflow Diagram

6.7 Examples

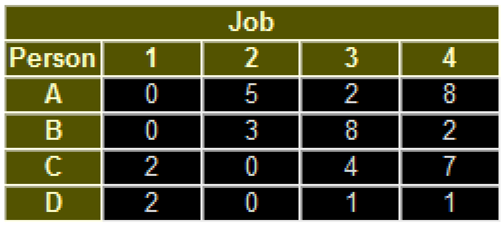
1. Consider the following example with the cost matrix as shown below:



	Job			
Person	1	2	3	4
A	20	25	22	28
B	15	18	23	17
C	19	17	21	24
D	25	23	24	24

Fig 6.6: Cost Matrix for Example 1

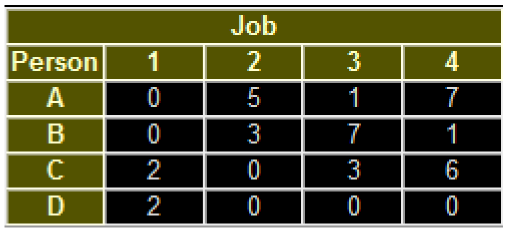
Step 1: The minimum element in every row is found out and subtracted from every element in that row.



	Job			
Person	1	2	3	4
A	0	5	2	8
B	0	3	8	2
C	2	0	4	7
D	2	0	1	1

Fig 6.7: Cost Matrix after Reduction on rows

Step 2: The minimum element in every column is found out and subtracted from every element in that column.



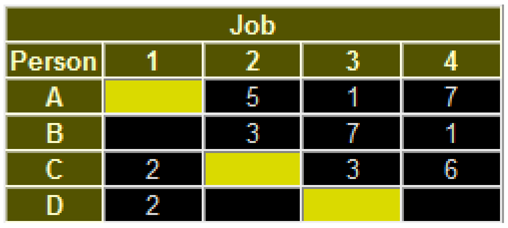
	Job			
Person	1	2	3	4
A	0	5	1	7
B	0	3	7	1
C	2	0	3	6
D	2	0	0	0

Fig 6.8: Cost Matrix after Reduction on columns

Step 3: The assignment is done on the matrix obtained after applying the first two steps in the following way:

i. For every row or a column which has only one zero which has not been assigned already or eliminated, we box that zero value and consider it as assigned.
ii. For every zero value that is assigned, we cross out all the zeroes present in the same row or column and mark them as eliminated.
iii. If for a particular row or a column, there are two or more zeroes and if we cannot choose one of them by inspection, then we can choose one zero arbitrarily and assign it.
iv. The above steps are executed until every zero in the matrix is either assigned or eliminated.

Step 4: If the number of assigned zeroes is equal to the number of people or jobs then an optimal assignment is found. If there are instances where zeroes have been chosen arbitrarily, then there may be different optimal solutions. If an optimal solution is not found then we go to step 5.



	Job			
Person	1	2	3	4
A		5	1	7
B		3	7	1
C	2		3	6
D	2			

Fig 6.9: Cost Matrix after initial assignment

Step 5: We have to now cover all the zeroes which are obtained through step 3 by using the following steps:

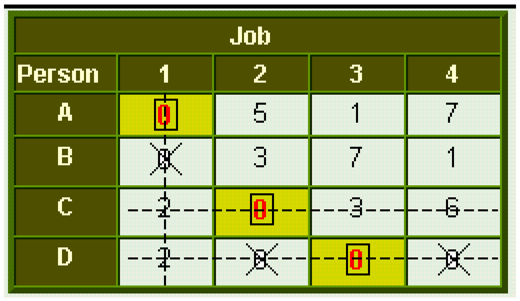
i. All the rows which are unassigned must be marked.

ii. Mark all the columns which have zeroes in the unassigned (marked) rows and which have not already been marked.

iii. Mark all the rows which are unmarked and which have assignments in the marked columns.

iv. Repeat the above three steps until no rows or columns can be marked.

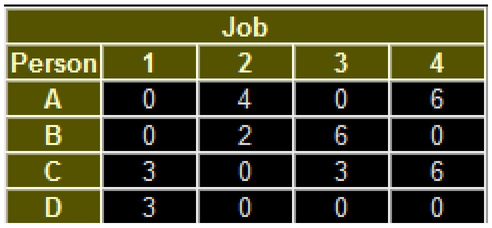
v. Draw lines through all unmarked rows and marked columns.



	Job			
Person	1	2	3	4
A	$\boxed{0}$	5	1	7
B	$\cancel{0}$	3	7	1
C	2	$\boxed{0}$	3	6
D	2	$\cancel{0}$	$\boxed{0}$	$\cancel{0}$

Fig 6.10: Cost Matrix after cancelling rows and columns with zeroes

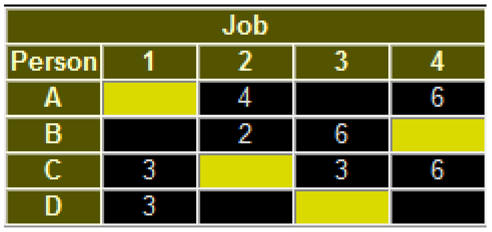
Step 6: Select the minimum element from all the elements through which a line has not been drawn. Subtract this minimum element from all the elements which do not have a line passing through them and add it to the elements which have two lines passing through them i.e. which have a line passing through the row as well as the column of that element.



	Job			
Person	1	2	3	4
A	0	4	0	6
B	0	2	6	0
C	3	0	3	6
D	3	0	0	0

Fig 6.11: Cost Matrix after new reduction

Now we have a new matrix on which assignment can be performed.



	Job			
Person	1	2	3	4
A	🟨	4		6
B		2	6	🟨
C	3	🟨	3	6
D	3		🟨	

Fig 6.12: Cost Matrix after final assignment

Since the number of assigned zeroes is equal to the number of people and jobs, this is the optimal solution. The total cost is: 20 + 17 + 17 + 24 = 78.

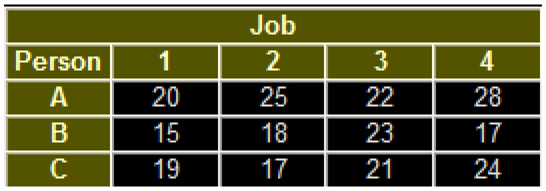
7. Special Cases

The assignment problem might encounter some special cases when specified by the user as an input. We discuss each of these special cases along with their solutions.

7.1 Unbalanced Assignment Problem

In this case the number of people and the number of jobs might not be equal. The Hungarian method is not directly applied in such a case. Dummy rows or columns with zero values are added as required in order to make the number of rows and columns equal.

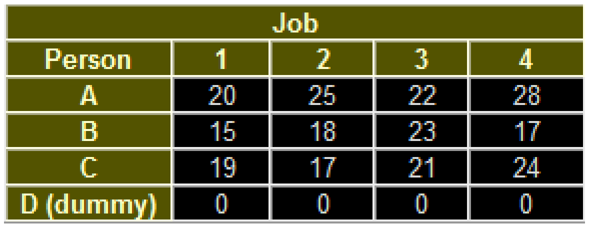
Consider the example below:



	Job			
Person	1	2	3	4
A	20	25	22	28
B	15	18	23	17
C	19	17	21	24

Fig 7.1: Cost Matrix for Unbalanced Assignment Problem

In the above example it can be seen that the number of people is less than the number of jobs. Hence we add an extra row with zeroes in order to make both the number of jobs and people equal to four is added. The following matrix is obtained:



	Job			
Person	1	2	3	4
A	20	25	22	28
B	15	18	23	17
C	19	17	21	24
D (dummy)	0	0	0	0

Fig 7.2: Cost Matrix for Unbalanced Assignment Problem after addition of dummy row

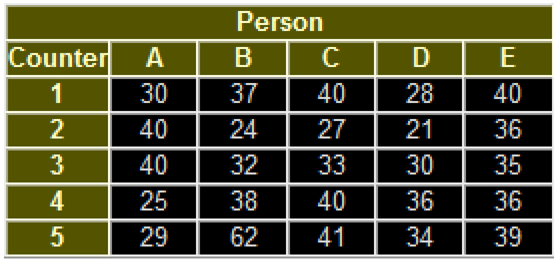
The optimal solution after applying Hungarian algorithm is as follows: 20 + 17 + 17 + 0 = 54.

7.2 Maximization in an Assignment Problem

There arise situations where certain properties have to be assigned to jobs such that the overall performance of the assignment is maximized. Such problems can be solved by the Hungarian algorithm by making some changes to the cost matrix before applying the algorithm. This change involves subtracting all the elements of the matrix from the maximum element and then applying the Hungarian algorithm to the resultant matrix. Finding the optimal solution for this matrix will be equivalent to finding the solution to the maximization problem.

Consider the example below:

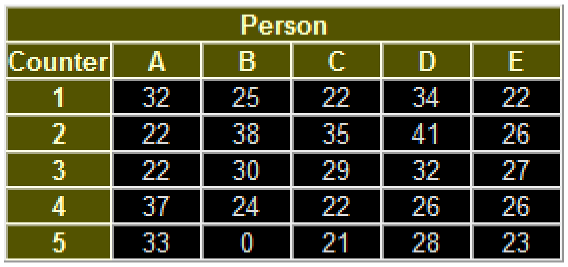
Here the counters have to be assigned to people such that the profit represented by the elements of the cost matrix must be maximized. The original cost matrix is as shown



	Person				
Counter	A	B	C	D	E
1	30	37	40	28	40
2	40	24	27	21	36
3	40	32	33	30	35
4	25	38	40	36	36
5	29	62	41	34	39

Fig 7.3: Cost Matrix for Maximization of assignment problem

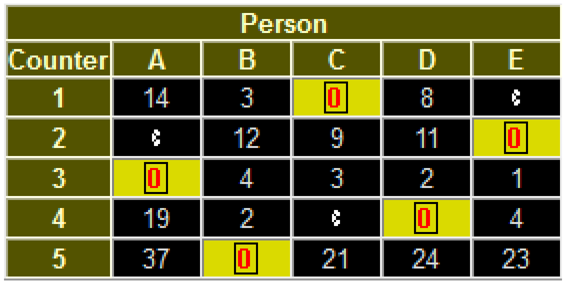
The highest value in the matrix is 62. Hence we subtract each and every value from 62.



	Person				
Counter	A	B	C	D	E
1	32	25	22	34	22
2	22	38	35	41	26
3	22	30	29	32	27
4	37	24	22	26	26
5	33	0	21	28	23

Fig 7.4: Cost Matrix after subtraction from maximum element

The Hungarian algorithm is applied on the above reduced matrix and the following solution is obtained.



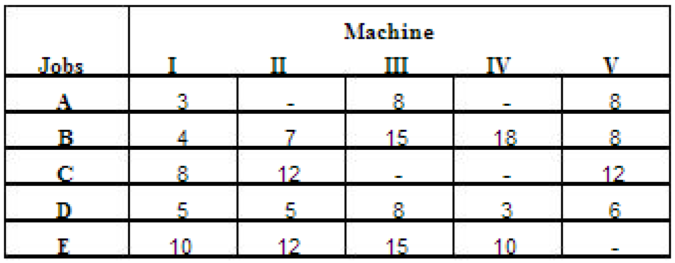
	Person				
Counter	A	B	C	D	E
1	14	3	$\boxed{0}$	8	$\cancel{0}$
2	$\cancel{0}$	12	9	11	$\boxed{0}$
3	$\boxed{0}$	4	3	2	1
4	19	2	$\cancel{0}$	$\boxed{0}$	4
5	37	$\boxed{0}$	21	24	23

Fig 7.5: Final Assignment

The total profit obtained by substituting the values in the original table is 40 + 36 + 40 + 36 + 62 = 214.

7.3 Unfeasible or Undesirable solutions

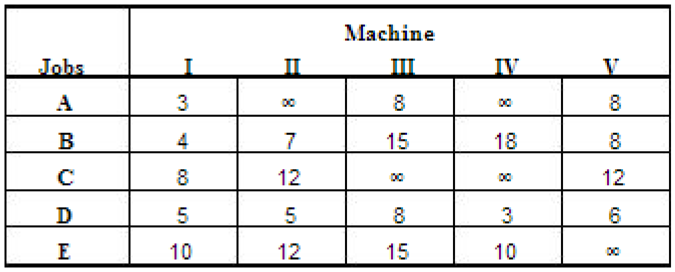
Sometimes some people or machines cannot perform some jobs. In such cases the input matrix consists of a '-'when a machine cannot perform a job. For such cases the value '-' is replaced by infinite or a very high value before applying the Hungarian algorithm so that these jobs are not assigned to those particular machines. Consider the following example where certain jobs cannot be performed by certain machines which is represented by '-'.



	Machine				
Jobs	I	II	III	IV	V
A	3	-	8	-	8
B	4	7	15	18	8
C	8	12	-	-	12
D	5	5	8	3	6
E	10	12	15	10	-

Fig 7.6: Cost Matrix to represent unfeasible or undesirable solutions

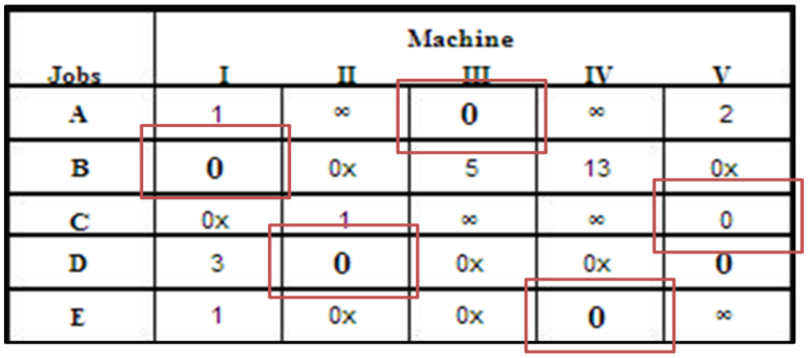
The '-' in the above input are replaced by infinite values in the below table and this table is used as input for the Hungarian algorithm.



	Machine				
Jobs	I	II	III	IV	V
A	3	∞	8	∞	8
B	4	7	15	18	8
C	8	12	∞	∞	12
D	5	5	8	3	6
E	10	12	15	10	∞

Fig 7.7: Cost Matrix using infinite to represent unfeasible or undesirable solutions

The solution obtained after the application of Hungarian algorithm is as follows:



Jobs	Machine I	II	III	IV	V
A	1	∞	**0**	∞	2
B	**0**	0x	5	13	0x
C	0x	1	∞	∞	0
D	3	**0**	0x	0x	**0**
E	1	0x	0x	**0**	∞

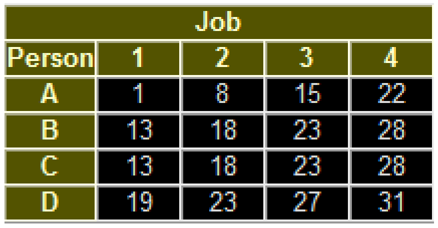
Fig 7.8: Cost Matrix after assignment

The total cost is 8+4+12+5+10=39.

7.4 Multiple optimal solutions

Sometimes a problem might have multiple optimal solutions i.e. two or more solutions which have the same sum. This case occurs when we select zeroes arbitrarily. These type of problems can be solved by using the backtracking method.

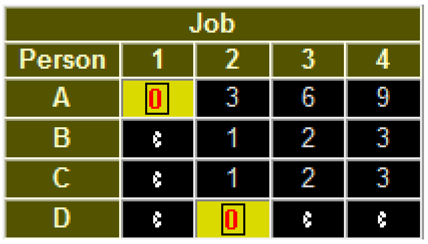
Let us consider the following example:



	Job			
Person	1	2	3	4
A	1	8	15	22
B	13	18	23	28
C	13	18	23	28
D	19	23	27	31

Fig 7.9: Problem with multiple optimal solutions

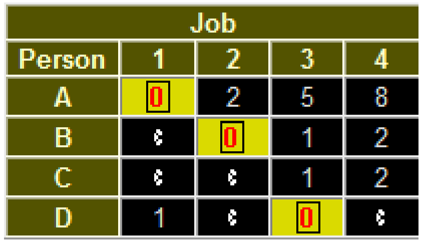
After the application of the first three steps of the Hungarian Algorithm, we obtain the following matrix:



	Job			
Person	1	2	3	4
A	0	3	6	9
B	¢	1	2	3
C	¢	1	2	3
D	¢	0	¢	¢

Fig 7.10: Cost matrix after application of first 3 steps of the Hungarian Algorithm

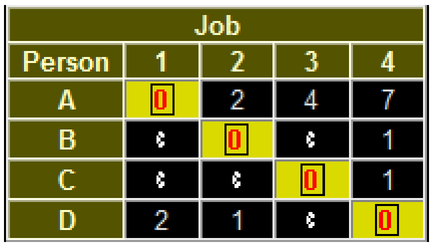
After applying the rest of the steps, we get:



	Job			
Person	1	2	3	4
A	0	2	5	8
B	¢	0	1	2
C	¢	¢	1	2
D	1	¢	0	¢

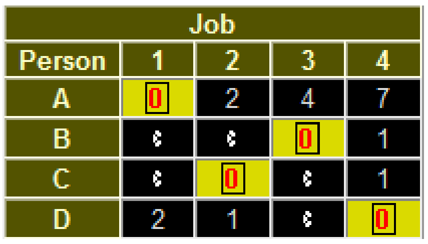
Fig 7.11: Cost matrix after application of the entire Hungarian Algorithm

There are two alternative assignments possible for this matrix which are:



	Job			
Person	1	2	3	4
A	0	2	4	7
B	¢	0	¢	1
C	¢	¢	0	1
D	2	1	¢	0

Sum: 1 + 18 + 23 + 31 = 73



	Job			
Person	1	2	3	4
A	0	2	4	7
B	¢	¢	0	1
C	¢	0	¢	1
D	2	1	¢	0

Sum: 1 + 23 + 18 + 31 = 73

Fig 7.12: Two assignments possible

8. User Interface

8.1 Input

The inputs required for the Hungarian algorithm to execute are:

- The number of people

- The number of jobs
- The cost matrix which specifies the cost that each person takes to complete each and every job
- The type of operation-maximize/minimize
- Any special constraints if present

We perform validation on the input provided because input in an incorrect format or an empty input can result in exceptions. Here the user provides three main inputs and we use three different functions to validate each of these inputs namely the number of people, number of jobs and the cost matrix. These functions are

The function validatetxtNoofJobs(); validates the input given in the text box used to take input for the number of jobs. If the user does not enter any input then the function returns 1 and the system prompts the user to enter the input. If the input is entered the function checks if only numerals are entered, if so the input is correct hence the function return 3 or else invalid characters have been entered as input and hence the value in the textbox is set to null and the function returns 2 and prompts the user to enter correct input.

The function validatetxtNoofPeople(); validates the input given in the text box used to take input for the number of people. If the user does not enter any input then the function returns 1 and the system prompts the user to enter the input. If the input is entered the function checks if only numerals are entered, if so the input is correct hence the function return 3 or else invalid characters have been entered as input and hence the value in the textbox is set to null and the function returns 2 and prompts the user to enter correct input.

The function validateTxtareaMatrix(); validates the input given in the text area used to take the cost matrix as input. If the user does not enter any input then the function returns 1 and the system prompts the user to enter the input or else it returns 3.

The function validateTextAreaInput(); validates the input textarea which takes the input cost matrix after the data is entered. This validation is performed just before performing the assignment (perform assignment function). It returns a string array if data entered if in correct format or else it returns null.

The function validateFileInput(); validates the content of the file browsed for input. The file must contain a sequence of numbers each separated by a space. It returns 0 if the file content is valid or else it returns 1.

The function validateFormat(); validates the restrictions entered by the user. The input must contain two numbers separated by a space. The function returns 0 if the file content is valid else it returns 1.

The function validateFileInputMatrixFormCheck(String line); checks whether the data in the file browsed is of correct form. It should be of format (number space number). This function is called in validateTextAreaInput for every line of input entered in the textarea. Hence the number of elements in each line of the textarea are equal to the number of jobs. This function returns 0 if input is valid or else it returns 1.

8.2 Output

The output is in the form of a table which shows which job has been assigned to which person. The sum of the optimal solution is also shown. If multiple optimal solutions are present, a table for each one of the solutions is displayed. The total sum is displayed on the title of the output dialog box.

9. Sreenshots

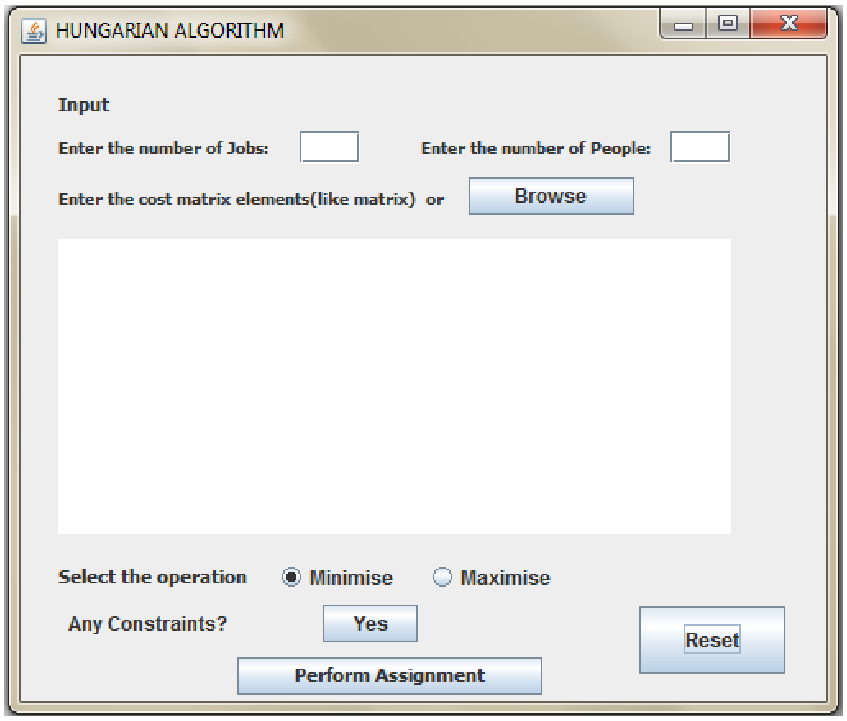
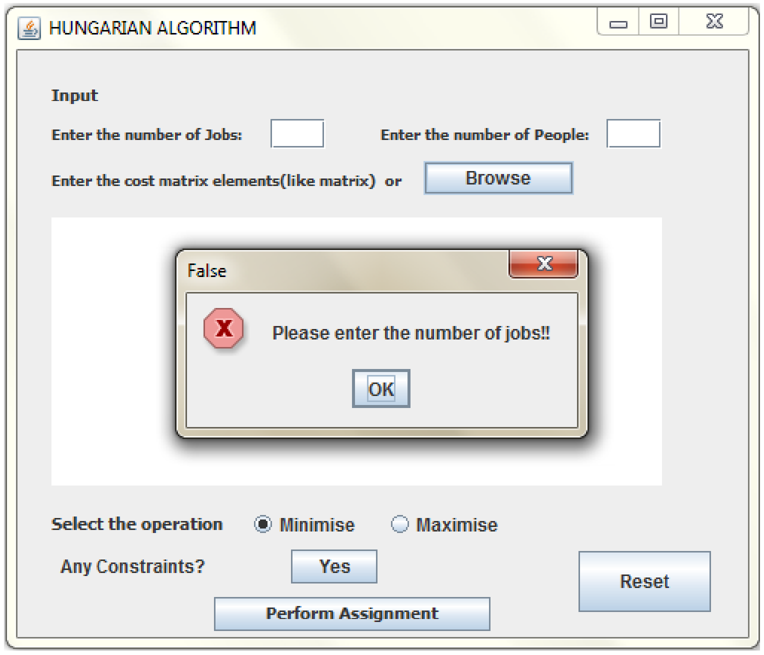
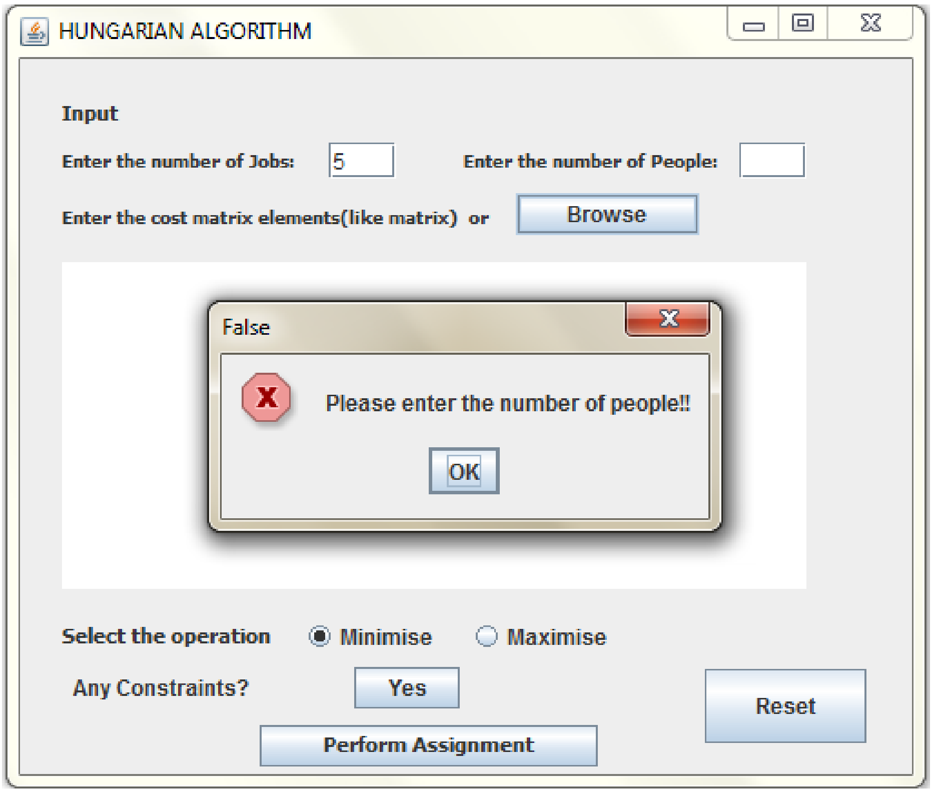


Fig 9.1: Input



Please enter the number of jobs!!

Fig 9.2: Validation for number of jobs


HUNGARIAN ALGORITHM

Input

Enter the number of Jobs: 5

Enter the number of People:

Enter the cost matrix elements(like matrix) or Browse

False

Please enter the number of people!!

OK

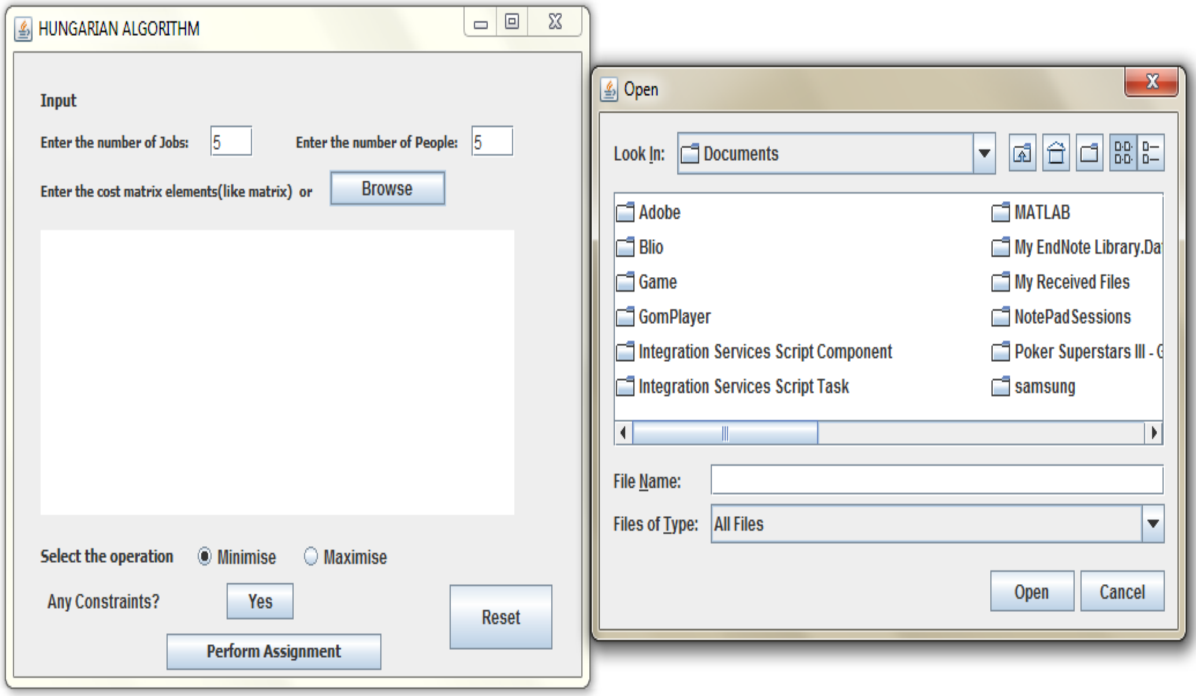
Select the operation [✓ Minimise] Maximise

Any Constraints? Yes

Perform Assignment

Reset

Fig 9.3: Validation for number of people


HUNGARIAN ALGORITHM

Input

Enter the number of Jobs: 5

Enter the number of People: 5

Enter the cost matrix elements(like matrix) or Browse

Select the operation [✓ Minimise] Maximise

Any Constraints? Yes

Perform Assignment

Reset

Open

Look In: Documents

Adobe
Blio
Game
GomPlayer
Integration Services Script Component
Integration Services Script Task
MATLAB
My EndNote Library.Da
My Received Files
NotePadSessions
Poker Superstars III - G
samsung

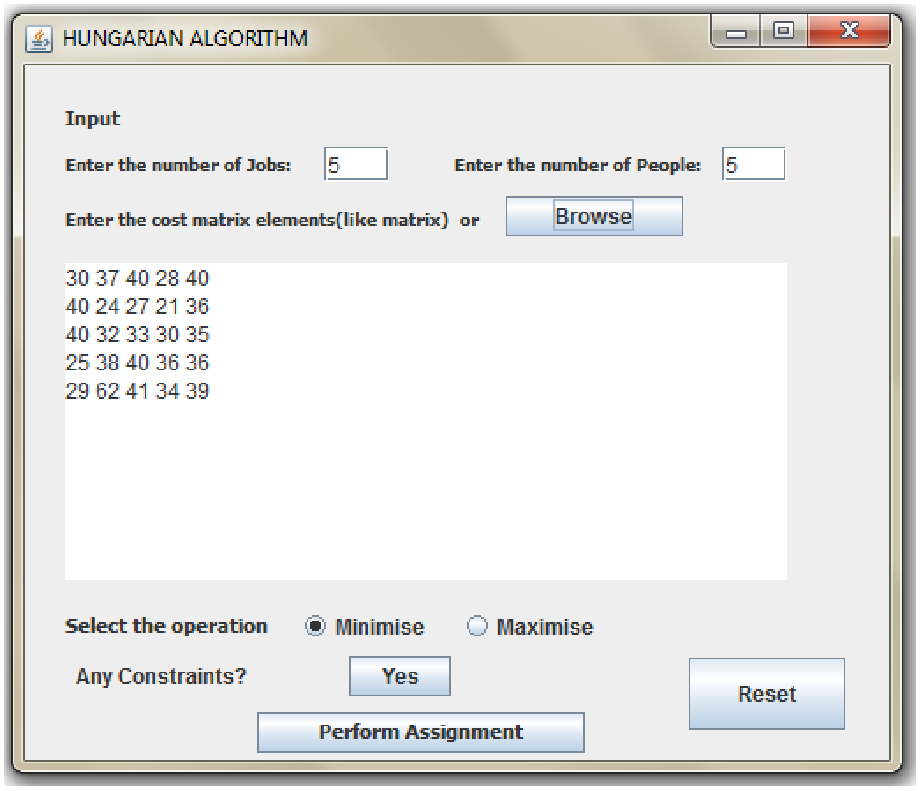
File Name:

Files of Type: All Files

Open Cancel

Fig 9.4: Browse File

The file browsed should contain the matrix elements separated by a space.



Enter the number of Jobs: 5

Enter the number of People: 5

Enter the cost matrix elements(like matrix) or

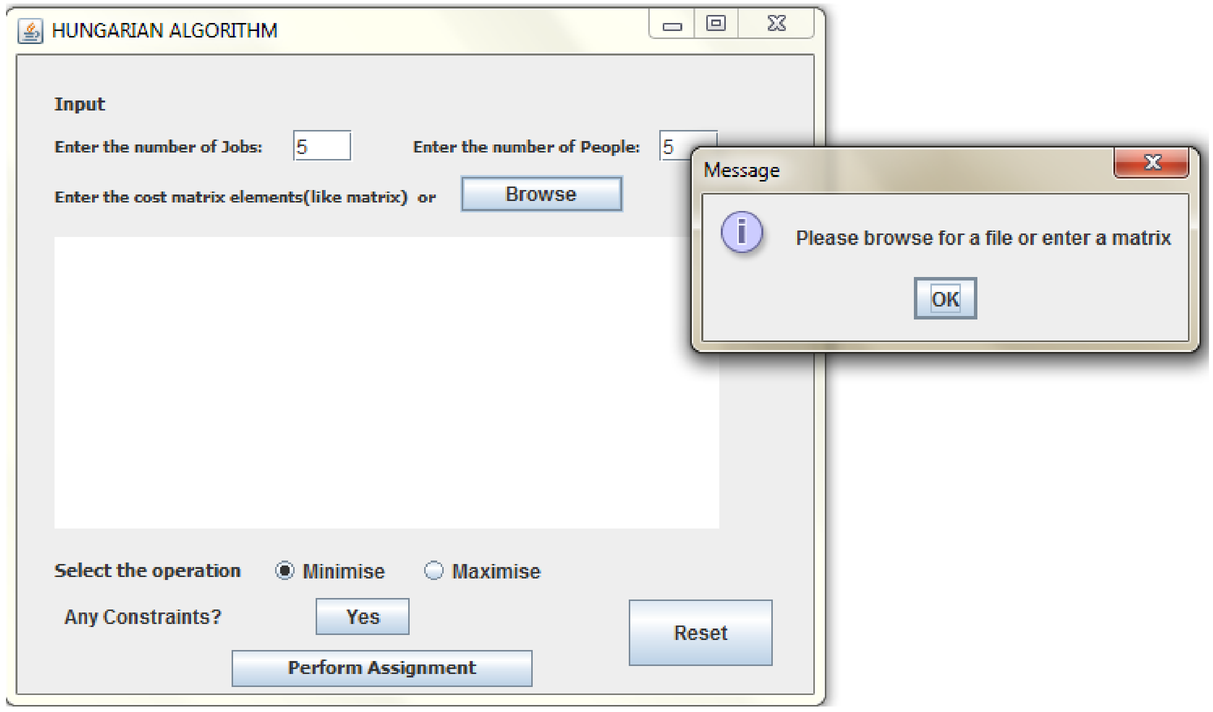
```
30 37 40 28 40
40 24 27 21 36
40 32 33 30 35
25 38 40 36 36
29 62 41 34 39
```

Select the operation [✓ Minimise] Maximise

Any Constraints?

Fig 9.5: Output of Browse File

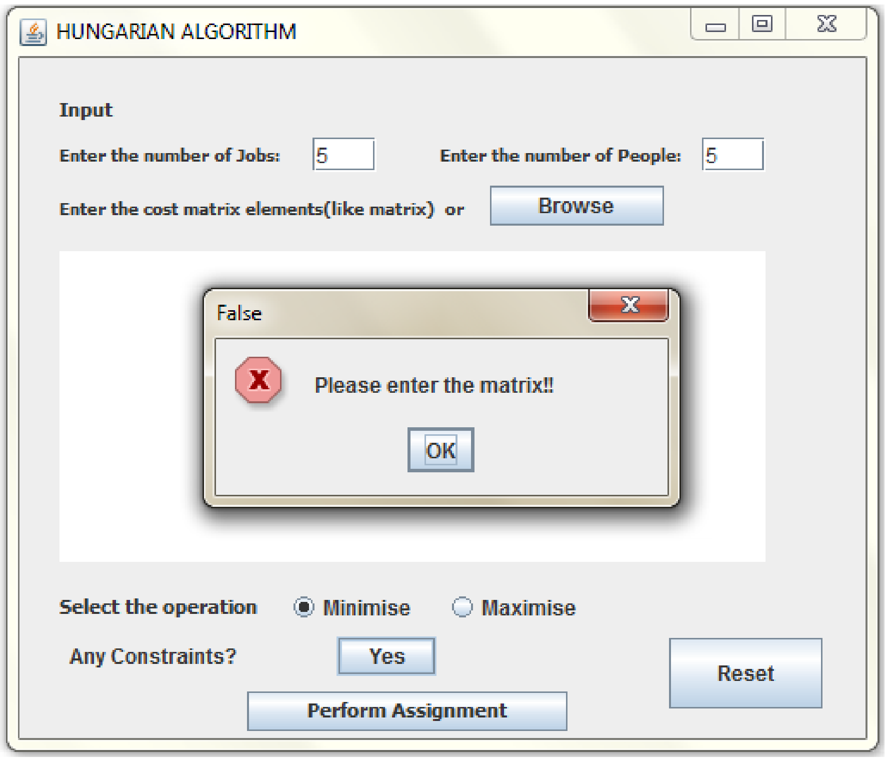
The below screenshot shows the validation message when user clicks on cancel button of the browse file dialog box.



Enter the number of Jobs: 5
Enter the number of People: 5
Select the operation [✓ Minimise] Maximise

Please browse for a file or enter a matrix

Fig 9.6: Validation for cost matrix



HUNGARIAN ALGORITHM

Input

Enter the number of Jobs: 5 Enter the number of People: 5

Enter the cost matrix elements(like matrix) or Browse

False

Please enter the matrix!!

OK

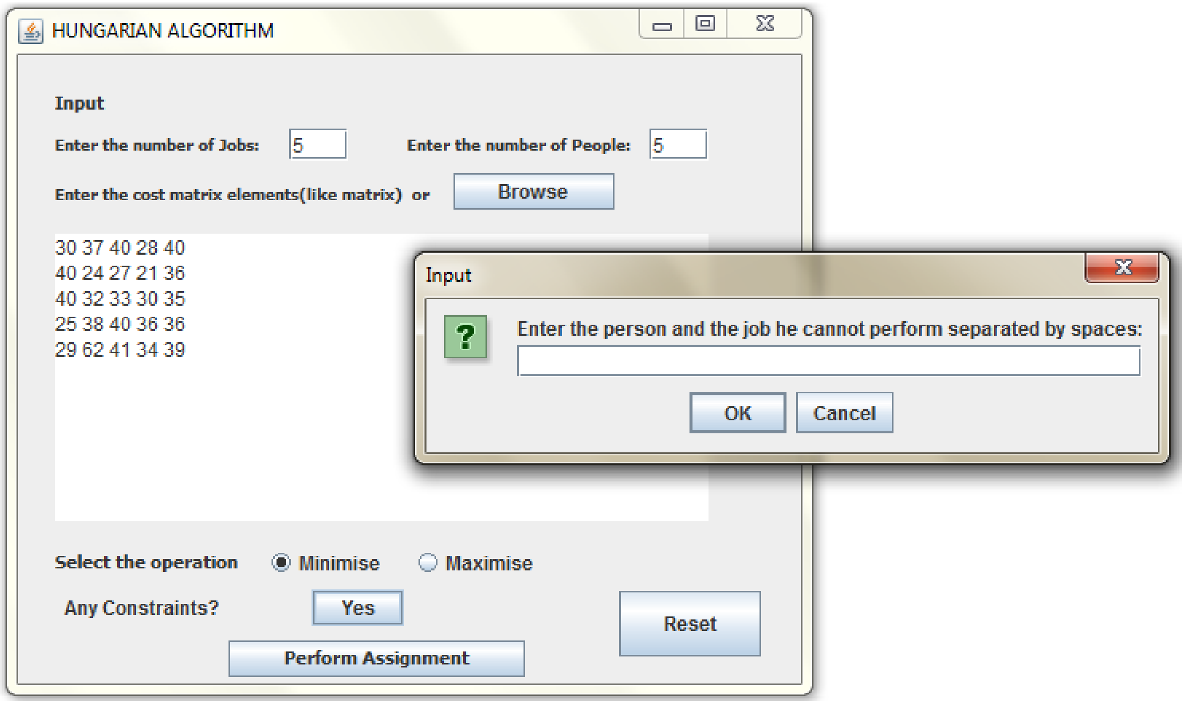
Select the operation [✓ Minimise] Maximise

Any Constraints? Yes

Perform Assignment

Reset

Fig 9.7: Validation for cost matrix before Perform Assignment



HUNGARIAN ALGORITHM

Input

Enter the number of Jobs: 5 Enter the number of People: 5

Enter the cost matrix elements(like matrix) or Browse

```
30 37 40 28 40
40 24 27 21 36
40 32 33 30 35
25 38 40 36 36
29 62 41 34 39
```

Input

Enter the person and the job he cannot perform separated by spaces:

OK Cancel

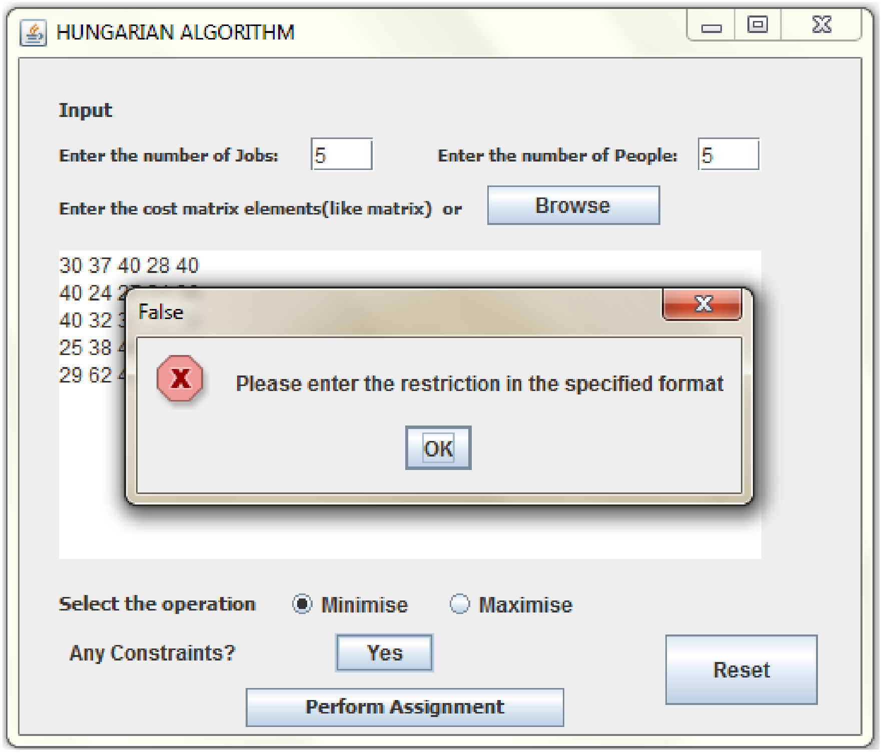
Select the operation [✓ Minimise] Maximise

Any Constraints? Yes

Perform Assignment

Reset

Fig 9.8: Prompt user for constraints



HUNGARIAN ALGORITHM

Input

Enter the number of Jobs: 5 Enter the number of People: 5

Enter the cost matrix elements(like matrix) or Browse

```
30 37 40 28 40
40 24 2
40 32 3
25 38 4
29 62 4
```

False

Please enter the restriction in the specified format

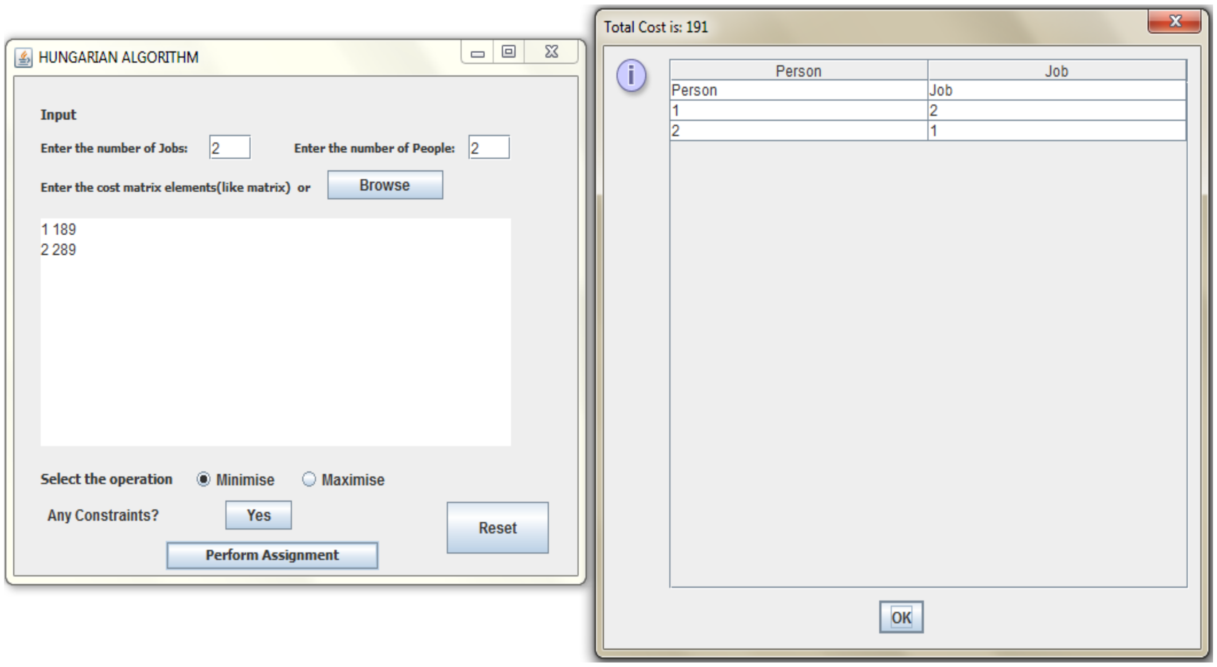
OK

Select the operation [✓ Minimise] Maximise

Any Constraints? Yes

Perform Assignment Reset

Fig 9.9: Validation of constraints



HUNGARIAN ALGORITHM

Input

Enter the number of Jobs: 2 Enter the number of People: 2

Enter the cost matrix elements(like matrix) or Browse

```
1 189
2 289
```

Select the operation [✓ Minimise] Maximise

Any Constraints? Yes

Perform Assignment Reset

Total Cost is: 191

Person	Job
Person	Job
1	2
2	1

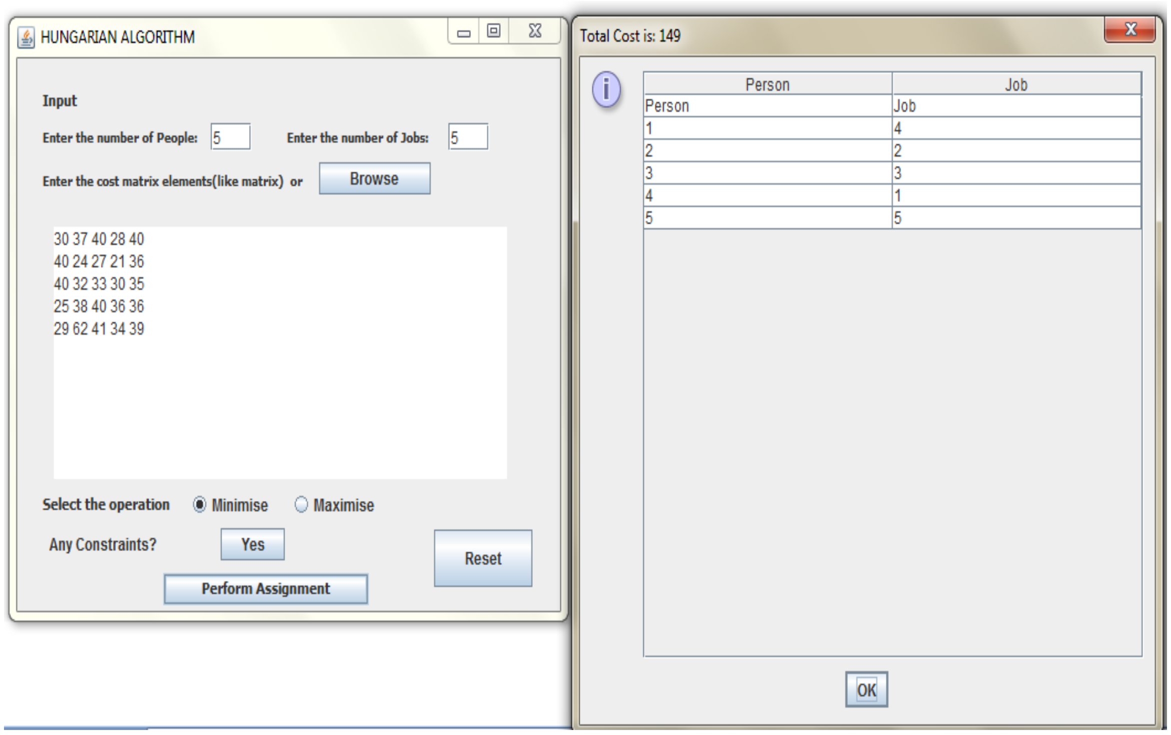
OK

Fig 9.10: Output of Optimal Solution

10. Testing examples

In this section we present the screen shots for the different examples and their corresponding outputs.

Example 1:



Enter the number of People: 5 Enter the number of Jobs: 5

Enter the cost matrix elements(like matrix) or

```
30 37 40 28 40
40 24 27 21 36
40 32 33 30 35
25 38 40 36 36
29 62 41 34 39
```

Select the operation [✓ Minimise] Maximise

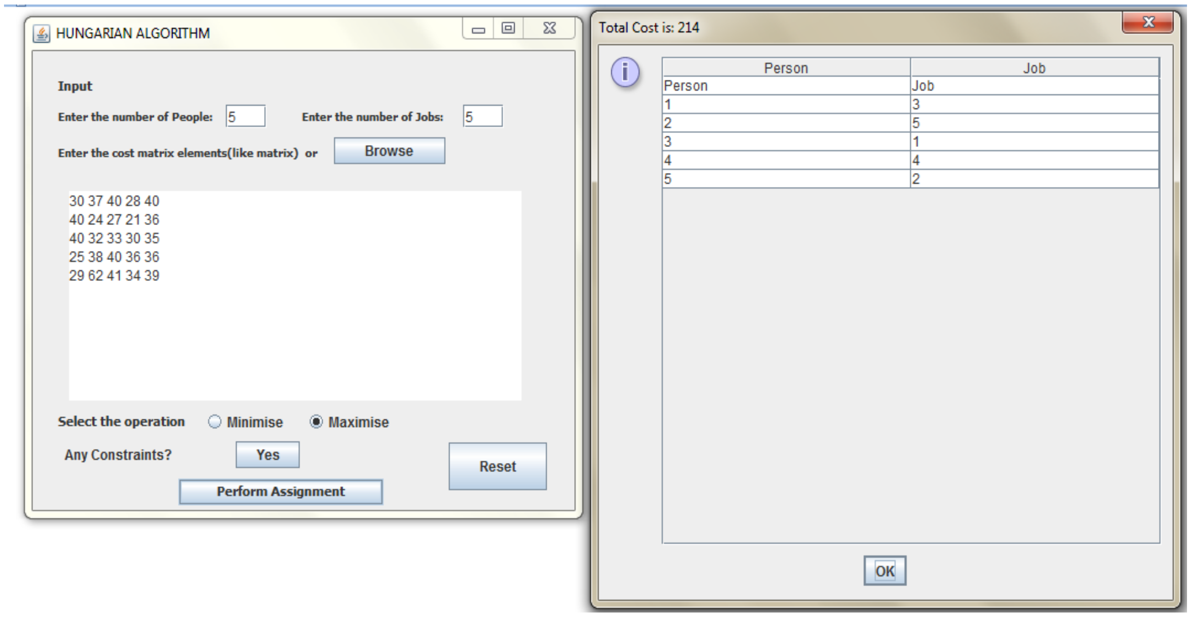
Any Constraints?

Total Cost is: 149

Person	Job
Person	Job
1	4
2	2
3	3
4	1
5	5

Fig 10.1: Output of Optimal Solution for example1

Example 2: This screen shot is for the maximization of the output. The input matrix for example 1 and 2 are same but the operation is different and this can be checked with the total sum displayed on the title of the output dialog box.



Enter the number of People: 5 Enter the number of Jobs: 5

Enter the cost matrix elements(like matrix) or

```
30 37 40 28 40
40 24 27 21 36
40 32 33 30 35
25 38 40 36 36
29 62 41 34 39
```

Select the operation ○ Minimise [✓ Maximise]

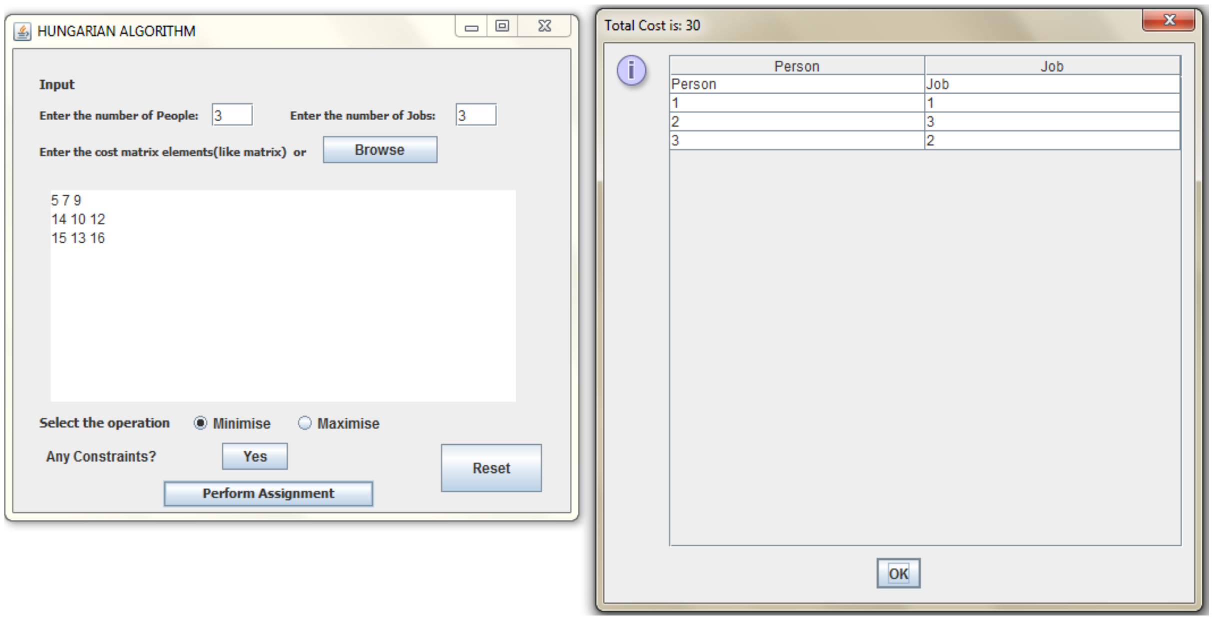
Any Constraints?

Total Cost is: 214

Person	Job
Person	Job
1	3
2	5
3	1
4	4
5	2

Fig 10.2: Output of Optimal Solution for Maximize for example2

Example 3. This screen shot is for 3 by 3 matrix input and its corresponding output.



Enter the number of People: 3 Enter the number of Jobs: 3

Enter the cost matrix elements(like matrix) or

```
5 7 9
14 10 12
15 13 16
```

Select the operation [✓ Minimise] ○ Maximise

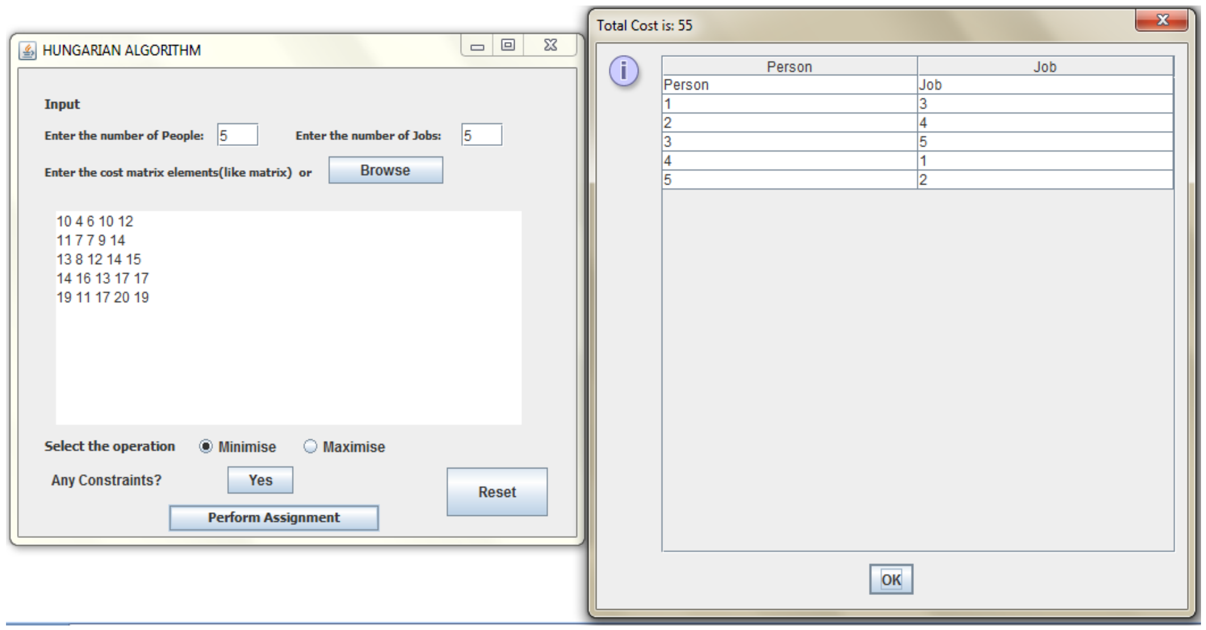
Any Constraints?

Total Cost is: 30

Person	Job
Person	Job
1	1
2	3
3	2

Fig 10.3: Output of Optimal Solution for example3

Example4



HUNGARIAN ALGORITHM

Input

Enter the number of People: 5 Enter the number of Jobs: 5

Enter the cost matrix elements(like matrix) or Browse

```
10 4 6 10 12
11 7 7 9 14
13 8 12 14 15
14 16 13 17 17
19 11 17 20 19
```

Select the operation [✓ Minimise] Maximise

Any Constraints? Yes

Perform Assignment

Reset

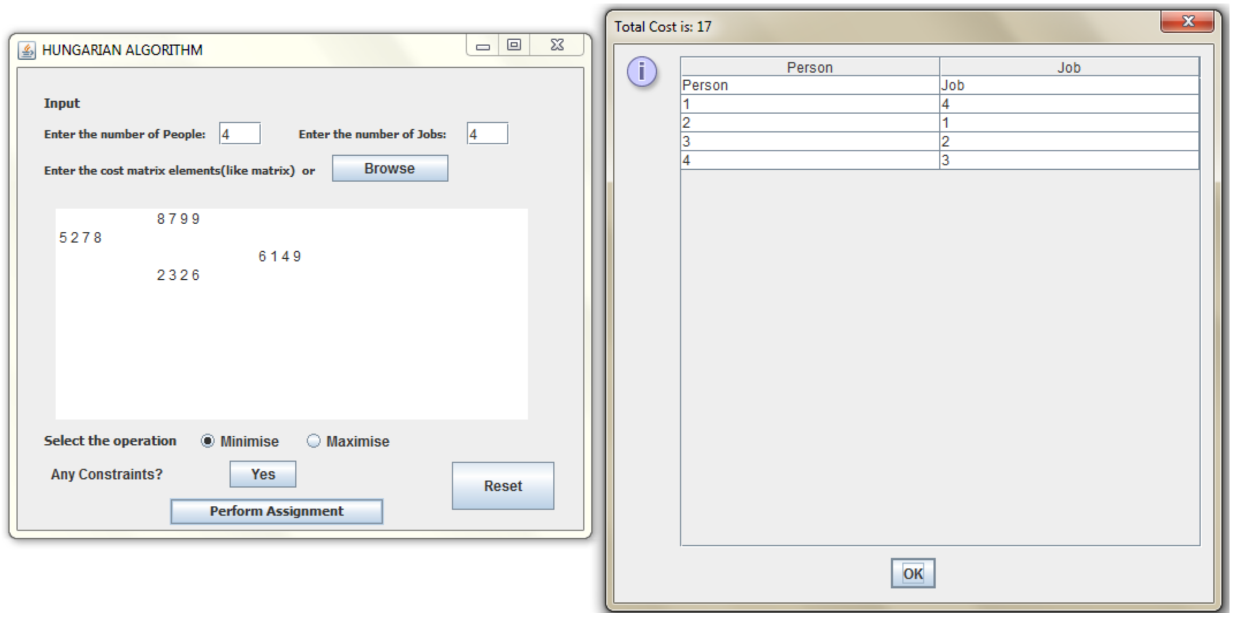
Total Cost is: 55

Person	Job
Person	Job
1	3
2	4
3	5
4	1
5	2

OK

Fig 10.4: Output of Optimal Solution for example4

Example 5. The input can entered as shown below but the elements should have a single space between them.



HUNGARIAN ALGORITHM

Input

Enter the number of People: 4 Enter the number of Jobs: 4

Enter the cost matrix elements(like matrix) or Browse

```
            8 7 9 9
5 2 7 8
                        6 1 4 9
            2 3 2 6
```

Select the operation [✓ Minimise] Maximise

Any Constraints? Yes

Perform Assignment

Reset

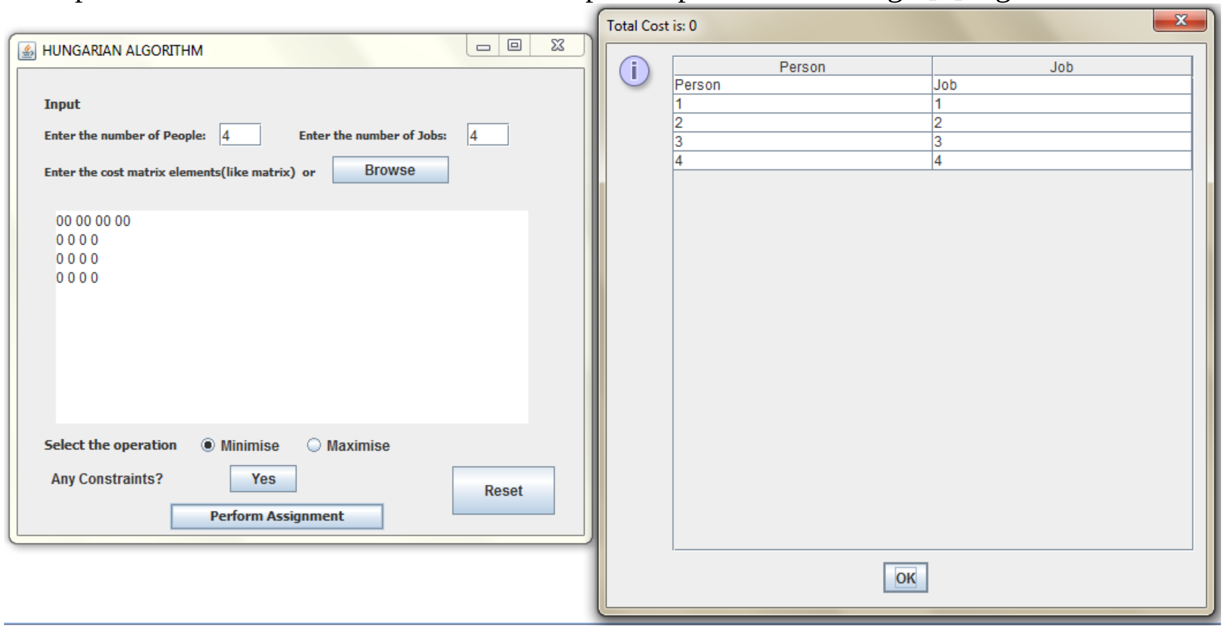
Total Cost is: 17

Person	Job
Person	Job
1	4
2	1
3	2
4	3

OK

Fig 10.5: Output of Optimal Solution for example5

Example 6. Screen shot for all zeros as the input. Implemented through [2] algorithm.



Enter the number of People: 4 Enter the number of Jobs: 4

Enter the cost matrix elements(like matrix) or

```
00 00 00 00
0 0 0 0
0 0 0 0
0 0 0 0
```

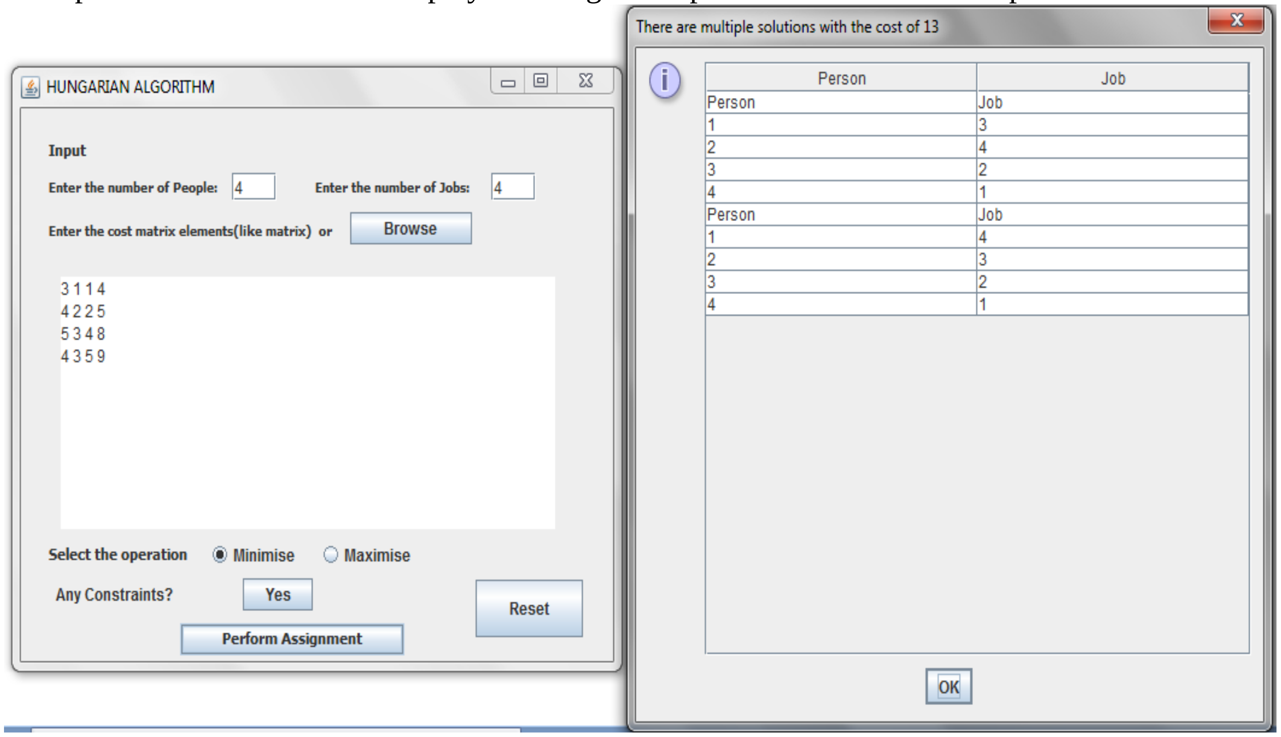
Select the operation: [✓ Minimise] Maximise

Total Cost is: 0

Person	Job
Person	Job
1	1
2	2
3	3
4	4

Fig 10.6: Output of Optimal Solution for example6

Example 7. This screen shots display the assignment problem which has multiple solutions.



Enter the number of People: 4 Enter the number of Jobs: 4

Enter the cost matrix elements(like matrix) or

```
3 1 1 4
4 2 2 5
5 3 4 8
4 3 5 9
```

Select the operation: [✓ Minimise] Maximise

There are multiple solutions with the cost of 13

Person	Job
Person	Job
1	3
2	4
3	2
4	1
Person	Job
1	4
2	3
3	2
4	1

Fig 10.7: Output of Optimal Solution for example7

11. References

1. H.W. Kuhn, The Hungarian Method for the Assignment Problem, Naval Research, Logistics Quarterly 2 (1955) 83–97
2. Munkres, J. Algorithms for the Assignment and Transportation Problems. Journal of the Society of Industrial and Applied Mathematics, 5(1):32-38, March, 1957.
3. Harold W. Kuhn. Variants of the Hungarian method for assignment problems. Naval Research Logistics Quarterly, 3: 253-258, 1956.