

# Project Report

A MAINTENANCE SCHEDULING  
PROBLEM

Aravinda Kumar Reddy Yempada  
ITCS 6150  
800877797

## Project Background and Description

Power system components are made to operate continuously throughout their life by means of preventive maintenance. Maintenance of a power unit requires the power outage of such a unit, which means some loss of the power capacity of the overall system and thus some loss in security. The security margin is determined by the system's net reserve, which is defined as the total installed generating capacity of the power system minus the power lost due to a scheduled outage and minus the maximum load forecast during the maintenance period. Maintenance scheduling is to find the sequence of outages of power units over a given period of time (normally a year) such that the security margin of the power system is maximized.

## Project Scope

Scope of this project is to use Constraint Satisfaction Problem and local search algorithm to solve the maintenance scheduling problem. The program should be able to deal with  $M$  power units and  $N < M$  maintenance intervals, with their information as input to the program and report the result. In addition, the project should include the report how the problem is formulated, the algorithm used, and the operator(s) and heuristics used to make a move, examples for testing, and the external documentation.

## Implementation Details

The problem is solved by combining Constraint Satisfaction Problem (CSP) and Local Search algorithm techniques. CSP is used to reduce the state space of the problem by imposing various constraints on the given problem. A local search algorithm is used to find the solution in the resultant state space using a heuristic cost criterion.

### Problem Approach:

The hill-climbing search algorithm is simply a loop that continually moves in the direction of increasing value—that is, uphill. It terminates when it reaches a "peak" where no neighbor has a higher value. The algorithm does not maintain a search tree, so the data structure for the current node need only record the state and the value of the objective function. Hill climbing does not look ahead beyond the immediate neighbors of the current state.

The pseudo code for hill climbing algorithm is given below:

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  current  $\leftarrow$  MAKE-NODE(problem.INITIAL-STATE)
  loop do
    neighbor  $\leftarrow$  a highest-valued successor of current
    if neighbor.VALUE  $\leq$  current.VALUE then return current.STATE
    current  $\leftarrow$  neighbor
```

Source: <http://aima.cs.berkeley.edu/algorithms.pdf>

Since, hill-climbing algorithm get stuck when it reaches a local maximum or plateau or ridge, random initialization is used whenever such scenario occurs.

For our problem, an instance of the problem is represented by a  $MXN$  matrix, where  $M$  is the number of units and  $N$  is the number of intervals.

Below are the steps to get the solution:

- Based on the given constraint (as the input array # intervals) that a unit should be scheduled in specified number of intervals, we generate the initial state for the hill-climbing problem randomly such that each unit is scheduled in exactly the same number of intervals that require to complete its maintenance.  
This way we will be able to reduce the solution space to much smaller space.
- Generate the successors of the initial state by changing the state of unit in one interval at a time.

- Each successor is validated if it satisfies the constraint, 'Net reserve must be greater or equal to zero'. If it does then it is added to the list of successors.
- Hill-climbing algorithm is applied on this list until a solution is reached based on a heuristic cost which is defined below.

The heuristic cost of our problem is based on the soft constraint, 'the net reserve must be at the maximum during any maintenance period'. As we always have the same number of units scheduled in every successor state, we have the same net reserve always. The maximum net reserve during any maintenance period can be given as below:

$$NR_{\text{max per interval}} = \text{Total Net Reserve} / \text{Number of intervals}$$

The heuristic cost for each successor state is evaluated as the sum of deviations of net reserve for each interval from the maximum possible net reserve in best case, which is given below:

$$\text{HCost} = \sum_{i=1}^{\text{Number of intervals}} \text{Abs}(\text{NetReserve in interval } i - NR_{\text{max per interval}})$$

### Problem Formulation:

Let us define the inputs of the problem:

- M is the number of units
- N is the number of intervals; where  $N < M$
- $C_i$  is the capacity of the  $i^{\text{th}}$  unit ; where  $i=1..M$
- $\text{Max}_i$  is the maximum load expected in an  $i^{\text{th}}$  interval; where  $i=1..N$
- An M sized IntervalsLeft array which has the number of intervals needed to complete the maintenance for each unit

The state of the problem can be represented by an  $M \times N$  array that has 0 if the unit is active and 1 if the unit is scheduled for maintenance.

- States: Any configuration of all the given units
- Initial state: Any state with all the intervals left for each unit is set to 1 can be designated as the initial state
- Actions: Changing the state of unit in one interval at a time
- Transition model: Based on the action, this results in a resulting state
- Goal test: A state which has the low heuristic cost

For the given example:

Interval	1	2	3	4
Max. loads (MW)	80	90	65	70

Unit Label	1	2	3	4	5	6	7
Capacity (MW)	20	15	35	40	15	15	10
# intervals	2	2	1	1	1	1	1

$M = 7$

$N = 4$

$C_1 = 20, C_2 = 15, C_3 = 35, C_4 = 40, C_5 = 15, C_6 = 15, C_7 = 10$

$Max_1 = 80, Max_2 = 90, Max_3 = 65, Max_4 = 70$

IntervalsLeft = [ 2 2 1 1 1 1 1 ]

An example initial state:

```

1 0 0 1
1 0 1 0
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
1 0 0 0

```

A successor state

0 1 0 1  
1 0 1 0  
1 0 0 0  
0 1 0 0  
0 0 1 0  
0 0 0 1  
1 0 0 0

A Goal State

0 1 0 1  
1 1 0 0  
0 0 0 1  
0 0 1 0  
0 0 1 0  
1 0 0 0  
1 0 0 0

References

- <http://aima.cs.berkeley.edu/algorithms.pdf>

## External Documentation

### Program Details

The program has been developed in C# using the Visual Studio IDE. It contains four .cs files

- Program.cs
- ProblemState.cs
- HillClimbing.cs
- Visualization.cs

Program.cs contains the Main() method from which the execution starts. Main() accepts the input from user and stores them in capacity, maxPerInterval and intervalsLeft arrays. It then creates an object of ProblemState class and generates an initial state. It then calls the HillClimbing method to find the solution. The solution is then displayed on the console and in a chart using the Visualization.cs file.

Following are the methods implemented in Program.cs file:

- GetInputFromConsole() – Handles the implementation of accepting the input from user. Uses CheckInput() to validate the input.
- GetInputFromFile() - Handles the implementation of accepting the input from a text file. The input file will be in the below format:

```
capacity : 20, 15, 35, 40, 15, 15,10,50,40,20
maxPerInterval : 80, 90,65,70,100
intervalsLeft : 2, 2, 1, 1, 1, 1, 1,3,1,1
```

- FindSolution() - Calls the HillClimbing method in HillClimbingProblem class.
- DisplayResult() – Displays the output on the console

The ProblemState class represents a state of the maintenance scheduling problem, it is implemented in the ProblemState.cs. ProblemState class contains the following properties and methods in it:

- Length – number of units, M
- NumberOfIntervals – number of intervals, N
- Solution – a MXN array that represents a state
- Capacity – a M sized array that contains the capacities of each unit
- maxPerInterval – a N sized array that contains the max loads of each interval
- intervalsLeft– a M sized array that contains the intervals needed to complete the maintenance of each interval
- total–the total capacity of all the units
- maxNetReservePerInterval– the maximum possible reserve per interval in the best case. Used to calculate the heuristic cost of the state
- ProblemState– Constructor to initialize the properties of the class with the user input. Also calculates the Total and maxNetReservePerInterval.
- InitializeState () – Method to randomly initialize a state
- GenerateNextStates () – Method that generates the successor states. It calls the Operations() method to generate each state. Calls the CheckIntervalConstraint() method to validate a state if it satisfies the constraints and stores the states in a list based on it
- HCost() – calculates the heuristic cost

HillClimbing.cs has the implementation of hill climbing approach to solve the n-queens problem. Hill-Climbing approach get stuck when it reaches a local maximum or plateau or ridge or if there are no successor states for a state, random initialization is used whenever such scenario occurs.

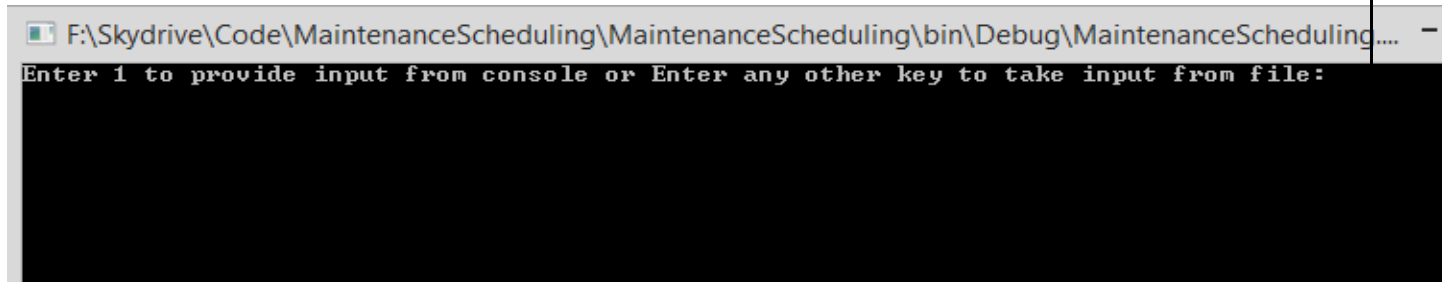
Visualization.cs has the implementation of displaying the output in graphical format.

Following two global variables are used in the program:

- numberOfInits – keeps track of number of random initializations
- numberOfSteps – keeps track of number of state changes



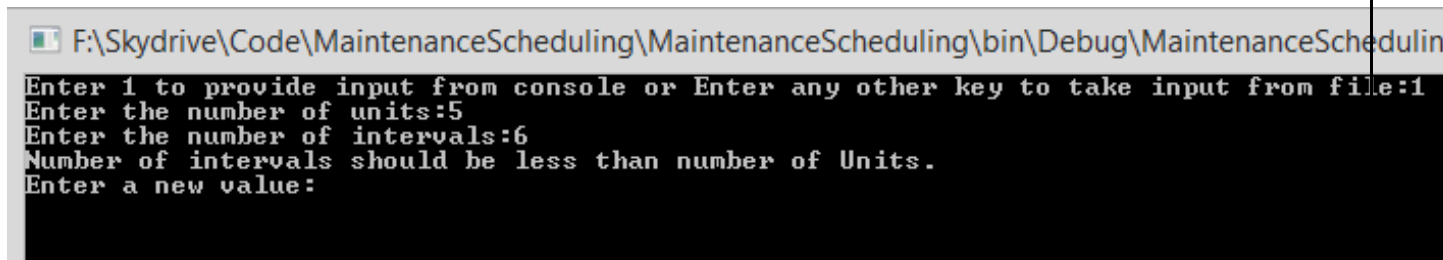
On execution, the program asks the user to Enter 1 to provide input from console or enter any other key to take input from file as shown below:



```
F:\Skydrive\Code\MaintenanceScheduling\MaintenanceScheduling\bin\Debug\MaintenanceScheduling.... -
Enter 1 to provide input from console or Enter any other key to take input from file:
```

If the user enters any other character except 1 then the program looks for the file “input.txt” in the execution directory, gets the input from the file and displays the output.

If the user enters 1, then the program asks for number of units, M and then the number of intervals, N. It does not accept N if it is greater than equal to M.



```
F:\Skydrive\Code\MaintenanceScheduling\MaintenanceScheduling\bin\Debug\MaintenanceScheduling.... -
Enter 1 to provide input from console or Enter any other key to take input from file:1
Enter the number of units:5
Enter the number of intervals:6
Number of intervals should be less than number of Units.
Enter a new value:
```

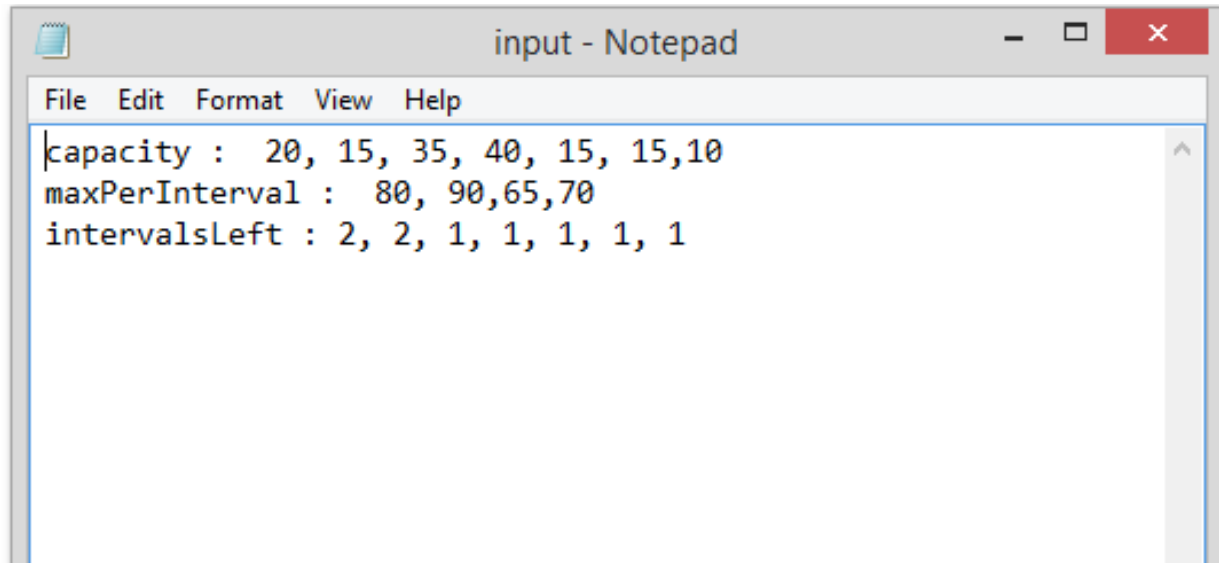
It then accepts the capacities of units, the maximum loads for each interval and number of intervals needed to complete a unit’s maintenance. Each units capacity should be greater than 0. Maximum load and intervals needed should be greater than or equal to zero. Other validations that are done are: maximum load should not exceed the total capacity and intervals needed should not exceed the total number of intervals N.

After getting the input, the program finds the solution and displays it on console along with the visualization.

## Execution Screenshots

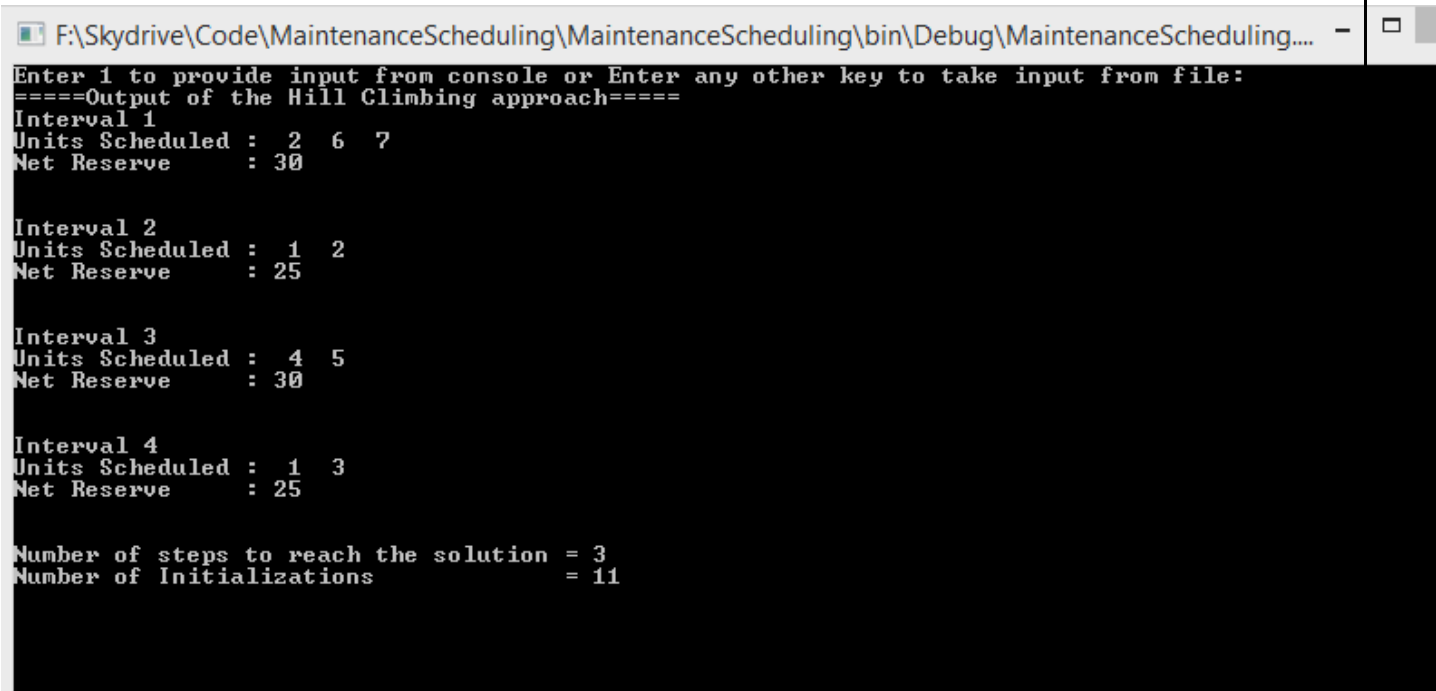
Example 1 – Given example

Input



```
File Edit Format View Help
capacity : 20, 15, 35, 40, 15, 15,10
maxPerInterval : 80, 90,65,70
intervalsLeft : 2, 2, 1, 1, 1, 1, 1
```

Output:



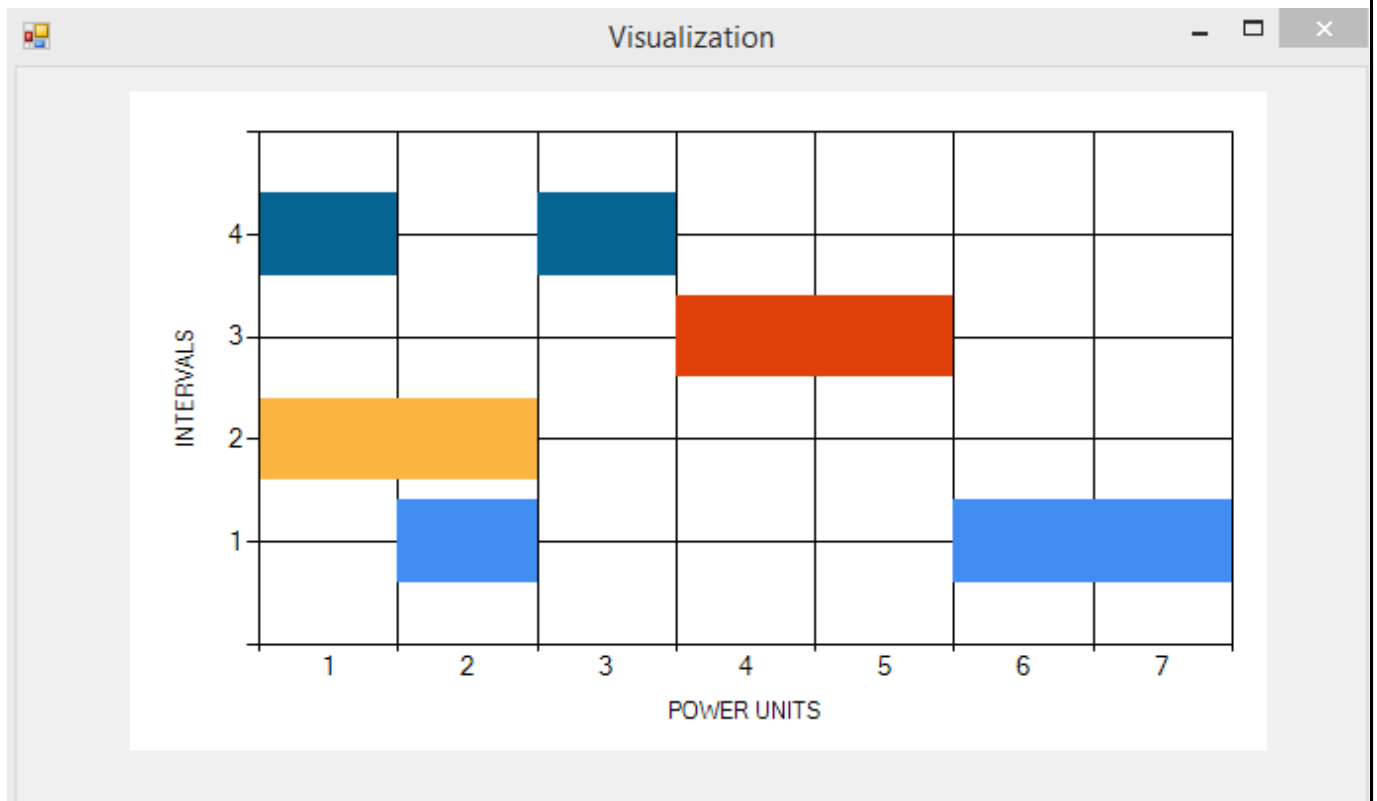
```
F:\Skydrive\Code\MaintenanceScheduling\MaintenanceScheduling\bin\Debug\MaintenanceScheduling....
Enter 1 to provide input from console or Enter any other key to take input from file:
=====Output of the Hill Climbing approach=====
Interval 1
Units Scheduled : 2 6 7
Net Reserve : 30

Interval 2
Units Scheduled : 1 2
Net Reserve : 25

Interval 3
Units Scheduled : 4 5
Net Reserve : 30

Interval 4
Units Scheduled : 1 3
Net Reserve : 25

Number of steps to reach the solution = 3
Number of Initializations = 11
```



## Example II

Input

```
capacity : 20, 15, 35, 40, 15, 15,10,50,40,20  
maxPerInterval : 80, 90,65,70,100  
intervalsLeft : 2, 2, 1, 1, 1, 1, 1,3,1,1
```

## Output

F:\Skydrive\Code\MaintenanceScheduling\MaintenanceScheduling\bin\Debug\MaintenanceScheduling

Enter 1 to provide input from console or Enter any other key to take input from file:

====Output of the Hill Climbing approach====

Interval 1

Units Scheduled : 1 6 8

Net Reserve : 95

Interval 2

Units Scheduled : 2 8

Net Reserve : 105

Interval 3

Units Scheduled : 4 8

Net Reserve : 105

Interval 4

Units Scheduled : 1 2 9 10

Net Reserve : 95

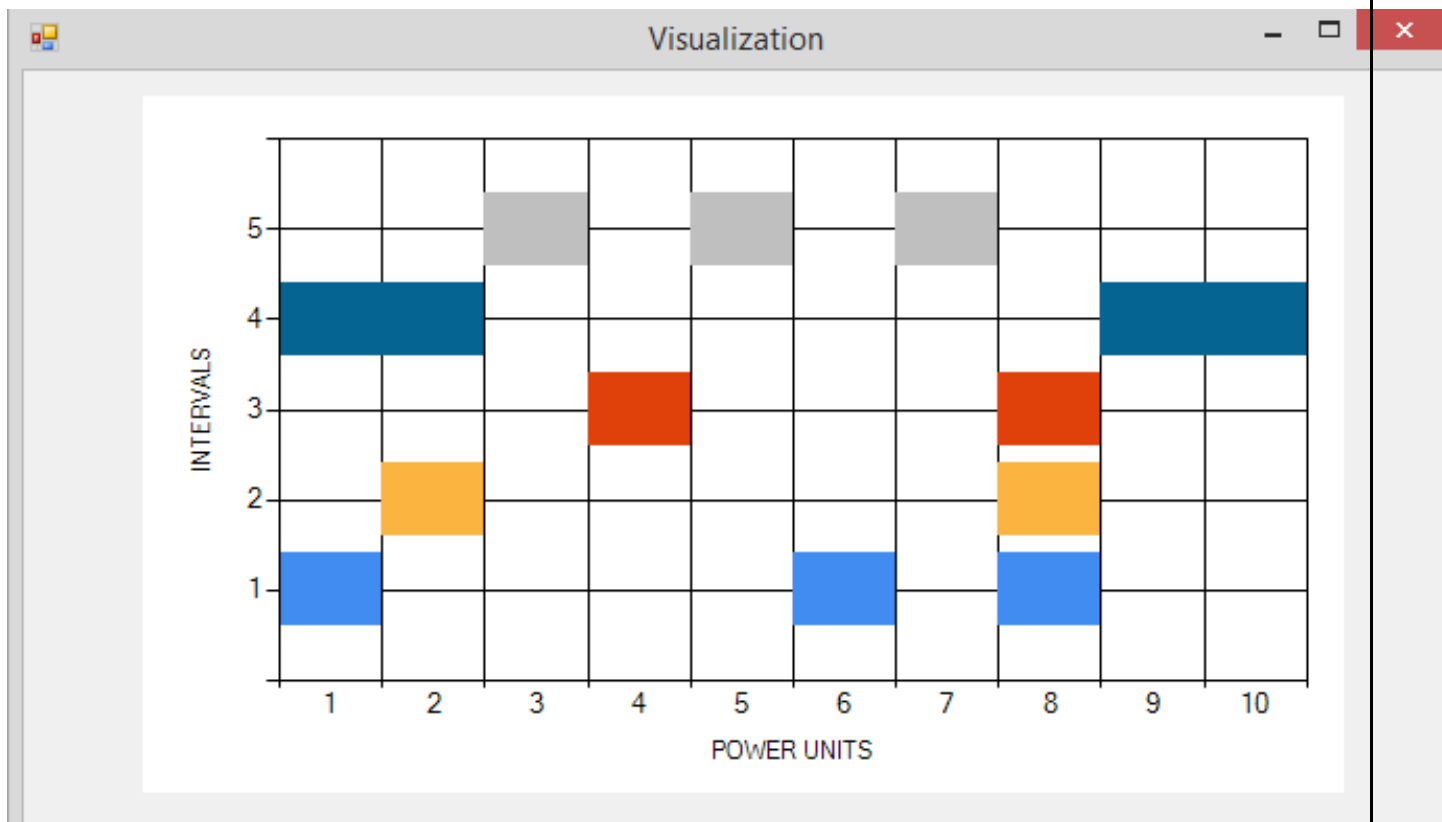
Interval 5

Units Scheduled : 3 5 7

Net Reserve : 100

Number of steps to reach the solution = 6

Number of Initializations = 3



### Example III

#### Giving Manual Input

F:\Skydrive\Code\MaintenanceScheduling\MaintenanceScheduling\bin\Debug\MaintenanceScheduling....

Enter 1 to provide input from console or Enter any other key to take input from file:1

Enter the number of units:6

Enter the number of intervals:4

Enter the Capacity of units

Unit 1:100

Unit 2:80

Unit 3:46

Unit 4:47

Unit 5:87

Unit 6:38

Enter the maximum loads expected during each interval

Interval 1:34

Interval 2:45

Interval 3:67

Interval 4:89

Enter the number of intervals required for each unit maintenance during a year:

Unit 1:2

Unit 2:1

Unit 3:1

Unit 4:1

Unit 5:1

Unit 6:2

====Output of the Hill Climbing approach====

Interval 1

Units Scheduled : 1 3

Net Reserve : 218

Interval 2

Units Scheduled : 2 4 6

Net Reserve : 188

Interval 3

Units Scheduled : 1 6

Net Reserve : 193

Interval 4

Units Scheduled : 5

Net Reserve : 222

Number of steps to reach the solution = 4

Number of Initializations = 1

