Dynamic Job Shop Scheduling with Sequence Dependent Routes using PSO

1. Sai Prasanna M S 2. Surbhi Agrawal
Dept of CSE , PESIT South Campus, Bangalore, Karnataka, India
(1)saiprasanna06@gmail.com, (2) surbhiagrawal@pes.edu

***Abstract*— Job shop scheduling (JSS) is a problem which involves an assignment of a set of tasks to the machines in a predefined sequence in order to optimize one or more objectives considering job performance measures of the system. It's a multi-objective, multi-criteria optimization problem. A job shop environment consists of 'n' jobs. Each job has a given machine routine in which some machines could be given a "miss". The JSS could be considered as a queuing system that consists of machines and jobs where each job demands a specified sequence (routing) on the machines and involves certain amount of processing time. In the dynamic JSSP, the problem size is not fixed. The jobs arrive randomly and addition of each job increases problem size exponentially. The optimal resource scheduling/re-scheduling for a mechanical manufacturing shop in reasonable computational time is achieved using particle Swarm Optimization (PSO). PSO (Particle swarm optimization) is a meta-heuristic method that calculates a combination of the sequence of the jobs yielding the least processing time or the make span (Best completion case)**

***Keywords*— DJSSP (Dynamic Job Shop Scheduling Problem), Particle Swarm Optimisation, Meta-heusistic, Optimization.**

I. INTRODUCTION

The machine Scheduling problem has been the one of the oldest problems of its kind. The inter-disciplinary nature of the problem expands its application in manufacturing systems, computer design, logistics etc. Among these, the traditional job shop scheduling problem is most frequently encountered.

The traditional JSSP has a fixed problem size, each job having a pre-defined sequence of machines and processing time. Under standard assumptions the processing time shall include the setup time as well. The problem is to find a solution that gives the minimum possible scheduling time for the given jobs. Since JSSP is one of the hardest combinatorial optimization problems, the solution attracts employing meta-heuristic methods because of its innate inflexible nature.

The Dynamic JSSP is a new method to break away from the static nature of the JSSP solution. Here the system system is able to accommodate a new job that needs to be added to the system at any random time. The complexity here is able to accommodate a new job that needs to be added to the system at any time. However, addition of a new job expands the problem size exponentially.

Some general rules of the JSSP that are followed in DJSSP are:

1. All jobs have equal priority
2. Every job visits the machine only once
3. No machine pre-emption is allowed
4. In a machine, only one job can be processed at once
5. All machines are available initially, at 0^{th} instance.

The solution proposed by various researchers, using Ant colony Optimisation [5] has not been able to handle the dynamic nature of the system efficiently.

We propose a Particle swarm optimization (PSO) based scheme to the dynamic job shop scheduling problem that will guarantee optimum processing time. Though it cannot assure optimal solution at all times due to the heuristic nature of the algorithm and the NP Hard nature of the problem, PSO is very promising. PSO [1] is based on the behaviour of social interaction and communication such as bird flocking and fish schooling. PSO does not employ the filtering operations which is the case in crossover or mutation. Nonetheless, it maintains communication in the population and keeps all the particles (Members of the population) informed. This enables the particles to frame themselves towards the best position in the search space.

In a PSO algorithm, each individual who works for the best fit is called particle. Velocity is a rate at which the particle is moving towards achieving the best fit. There are two criteria based on which the best fit can be measured. Global Solution is the best fit obtained so far by any particle in the population. Local solution is the best fit obtained so far by the particles in the neighbourhood.

The nature of arrival of a new job to the system is random. Our work proposes a system to handle this using PSO.

II. RELATED WORK

Kennedy, J.; Eberhart, R [1], proposed the conventional Particle Swarm Optimization method for optimization of non-linear functions.

This was further applied to the traditional JSSP by several methods [2][3]. However PSO applied on some variations of JSSP had different goals such as optimization of multi-objective flexible JSSP, flow shop sequencing, minimum make span etc. The method which comes close is the Ant Colony Optimization (ACO). However the comparison of the PSO with ACO makes us realize the following drawbacks of ACO [4].

1. There are some difficulties in the theoretical analysis

2. The Probability Distribution changes with each iteration

3. The time to reach the best solution is uncertain.

Thus the traditional PSO is more suitable to address this problem. Though there are many variants in PSO, the traditional PSO serves the purpose efficiently.

Swarm Intelligence (SI) is another important optimization method [6] [7] in evolutionary computation. SI is the property of a system whereby the collective behaviour of (unsophisticated) agents interacting locally with their environment cause coherent functional global patterns to emerge.

III. PROPOSED WORK

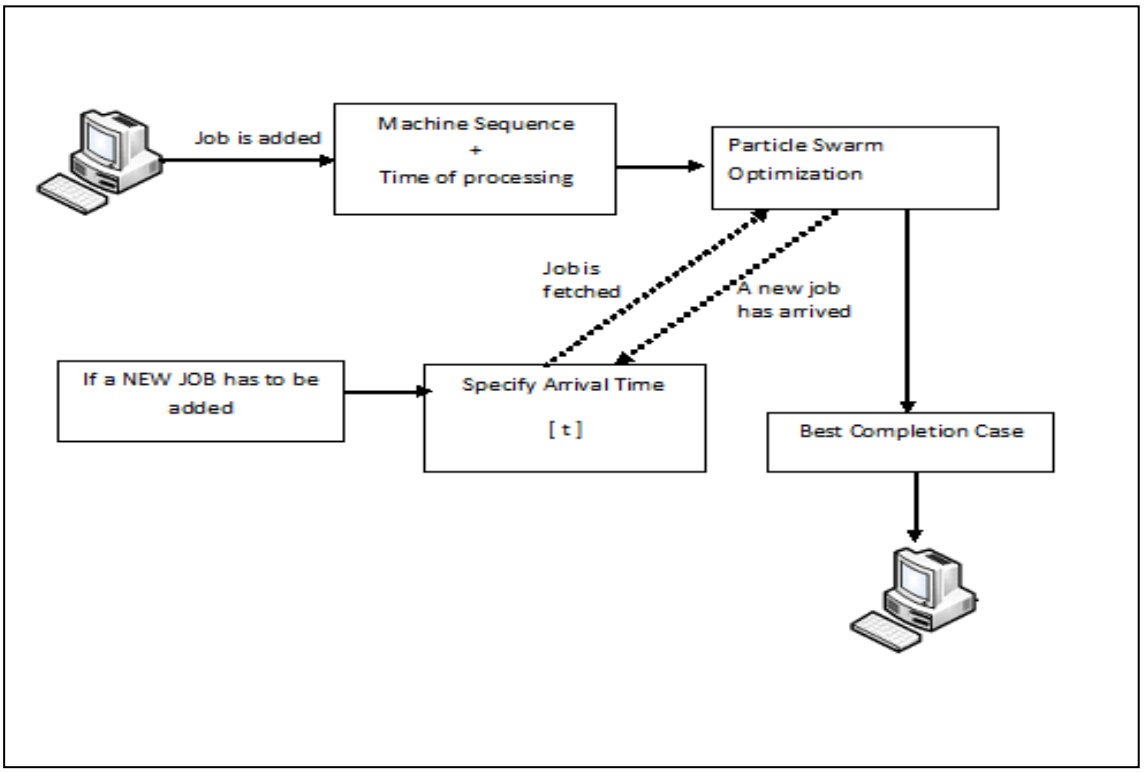

Figure 1 : Proposed Architecture

Suppose that the search space is D-dimensional and there are p particles in the swarm. Particle i is located at position

$$X_i = \{x_{1i}, x_{2i}, x_{3i} \dots x_{Di}\}$$

and has velocity

$$V_i = \{v_{1i}, v_{2i}, v_{3i} \dots v_{Di}\}$$

where i=1, 2, ..., p

As suggested by the PSO algorithm, each particle move towards its own best position (pbest), denoted as

$$P_{besti} = \{p_{best1i}, \quad p_{best2i}, \dots p_{bestni}\} \quad \text{..Eq (1)}$$

and the best position of the whole swarm (gbest) is denoted as

$G_{best_i} = \{g_{best1i}, g_{best2i}, \cdots g_{bestn}\}$ … Eq (2)

And gets updated after each iteration. Each particle changes its position according to its velocity, which is randomly generated toward the pbest and gbest positions.

This paper addresses the dynamic nature which is a variant in the traditional JSSP. The traditional PSO is applied on to the set of job(s) that arrive to the system at 0^{th} time instance when all the machines are available.

The problem is formulated as:

n is the total number of jobs

m is the total number of machines.

Typically the m*n matrix defines the problem size.

t(i,j) denotes the processing time of job i on machine j,

where i=1,2,3,…n j=1,2,3,4,…n

$\{p_1, p_2, \ldots p_n\}$ denotes the permutation of jobs.

The objective of this work is to find an optimal makespan M(p,j)

$M(p_1, 1) = t(p_1, 1)$ … Eq(3)

$M(p_1, 1) = M(p_{i-1}, 1) + t(p_1, 1)\ i = 1,2,3 \ldots n$ Eq (4)

$M(p_1, j) = c\ (p_i, j-1) + t(p, j)\ where\ j = 2 \ldots . m$ Eq(5)

Therefore, Makespan $f_{c\,max} = M(p_n, m)$ … Eq(6)

This is the stated objective function.

Any job can be represented as

$$Am_{(1\_n)} + Bm_{(1\_n)} + Cm_{(1\_n)} + \cdots + Nm_{(1\_n)} = 1$$

Eq (7)

A, B, C represent the time at machine m_1, m2, m3…m_n

The order of the machines for each job is fixed. Hence it cannot be optimal if we try re-sequencing the jobs and the machine sequence. We shall keep the job as a whole and not try breaking it up. We only consider the jobs converge to an optimal make span with less tardiness.

Stage 1: When there are no jobs in the system. (0^{th} Instance)

All machines are available.

Initialize the following parameters:

The search space

Number of particles (The population size)

The random numbers, S1, S2

The stopping criterion:

- Maximum number of iterations
- Stall point – Number of iteration the system waits for convergence at a single value.

i.e. let us assume that Stall point=10. If there are no changes in the solution in consecutive 10 iterations, then that solution is considered as final and the algorithm gets terminated.

Do
For every particle
Find the fitness value (The solution is able to calculate)
If the obtained fitness value is higher than the best fitness previously obtained
set current value as the new Local-Solution.
End

Choose the particle with the best fitness value of all the particles as the Global-Solution
For each particle
Calculate velocity using the following equation:
Update the Particle solution using the following equations:

$$v = v + S1 * rand()\ (localSolution - CurrentSolution) + S2 * rand() * (Global\ Solution - Current\ Solution)$$

… Eq (8)

$$Current\ Solution = Current\ Solution + v$$ … Eq (9)

End

Where v is the particle velocity, current Solution is the solution available for a particle at that instance. Rand () is a random number between (0,1).

S1, S2 are the constants

Stage 2: Handling randomly arriving jobs

The system is aware of all the existing Jobs and the corresponding machine queue status. Since the pre-emption of machine is not allowed, it is essential that we do not take away the jobs that are already allotted to the machine. However the jobs outside the machine but

allotted in the machine queue may be considered for re-sequencing. As stated earlier, the objective is to find the best sequence considering all the machines that give us the best make-span.

To handle the new job that arrives, from the available list of Job-Machine combination, freeze all the machine allocations made until time t=arrival time of the new job.

1. For the new schedule, rule out all the Job-Machine combinations existing in the queue before time t

2. For every machine, check if there is any lag that appears beyond time t. If yes, set the available time of that machine as t+t1, t1 being the time lag existing from the previous schedule

3. Discard all the schedules made after time=t in the previous stage

4. Check the availability time of all the machines and update the heuristics

5. Run the stage 1 method with starting time=t, thus obtaining the solution.

Step 1 deals with the "no-pre-emption rule" i.e. the jobs that are already getting processed in a machine cannot be taken away until they are completed.

In step-2, since the system maintains the queue status of all machines, it also has details of the "availability time index" of the machines. If a job under processing takes time>t, then the available time shall be set based on the time lag.

In step 3, apart from all the jobs that are currently occupying the machine at time=t (not the ones in the respective machine queues), we dissolve all other schedules made and re-consider them.

In step 4, we update the "availability time index" of all the machines according to the PSO algorithm which helps us in find the new sequence.

Step 5 is the repetition of stage 1, with the input as the set of job after time=t and the starting time is set as t.

Stage 2 Algorithm

```
Do
  Set Arrival time=t
 If
   Mi ∈ M(1....n) holding any job at time=t
   Lock the schedule
   Set available time t1=end time of current job
 For every machine M(1...n) - Mi
        Set machine availability time=t
  End
 End
For every Machine Queue
   For every job
   If  Scheduled time of process > t
      Discard the schedule
   End
  End
End
Update parameter 1 p1: Time of availability of every machine
Update parameter 2 p2: Time of arrival of new job
Update Parameter 3 p3:  List of Jobs to be processed
Stage 1 (p1,p2,p3)
End
```

IV. RESULTS AND DISCUSSIONS

https://www.youtube.com/watch?v=yV2g_TUHFTw&feature=youtu.be

The visual demo re-iterates the theoretical prediction that the system is capable of performing as per the requirements. The results have proved that the system is robust, scalable and has very less computational complexity.The following snap shots provide a deeper insight.

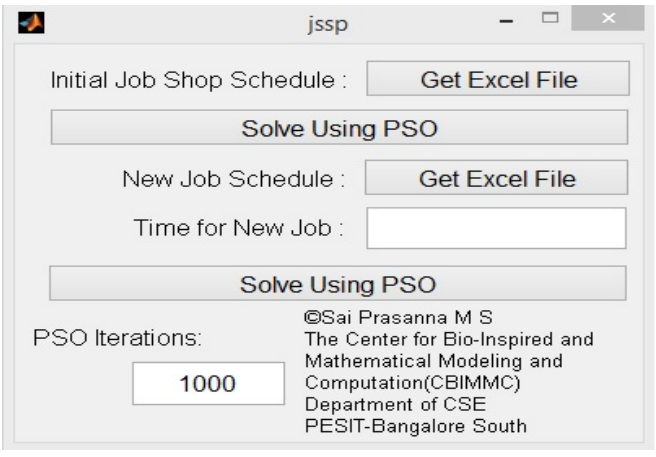

Fig 2 : The Graphical User Interface

Fig 4 : The corresponding Time Matrix

Case Study 1

Stage 1: 0th Instance of Time

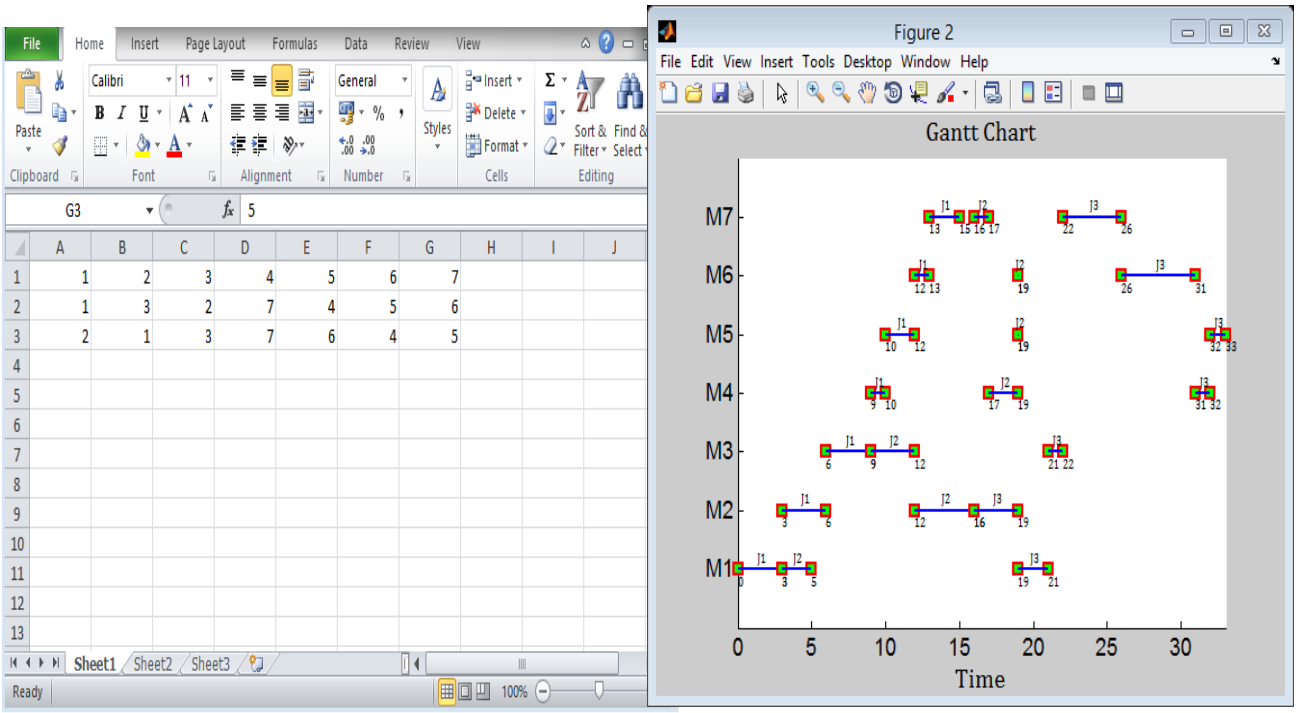

Fig 5 : Output for the Input at time=0

Fig 3 : The Machine sequence Matrix

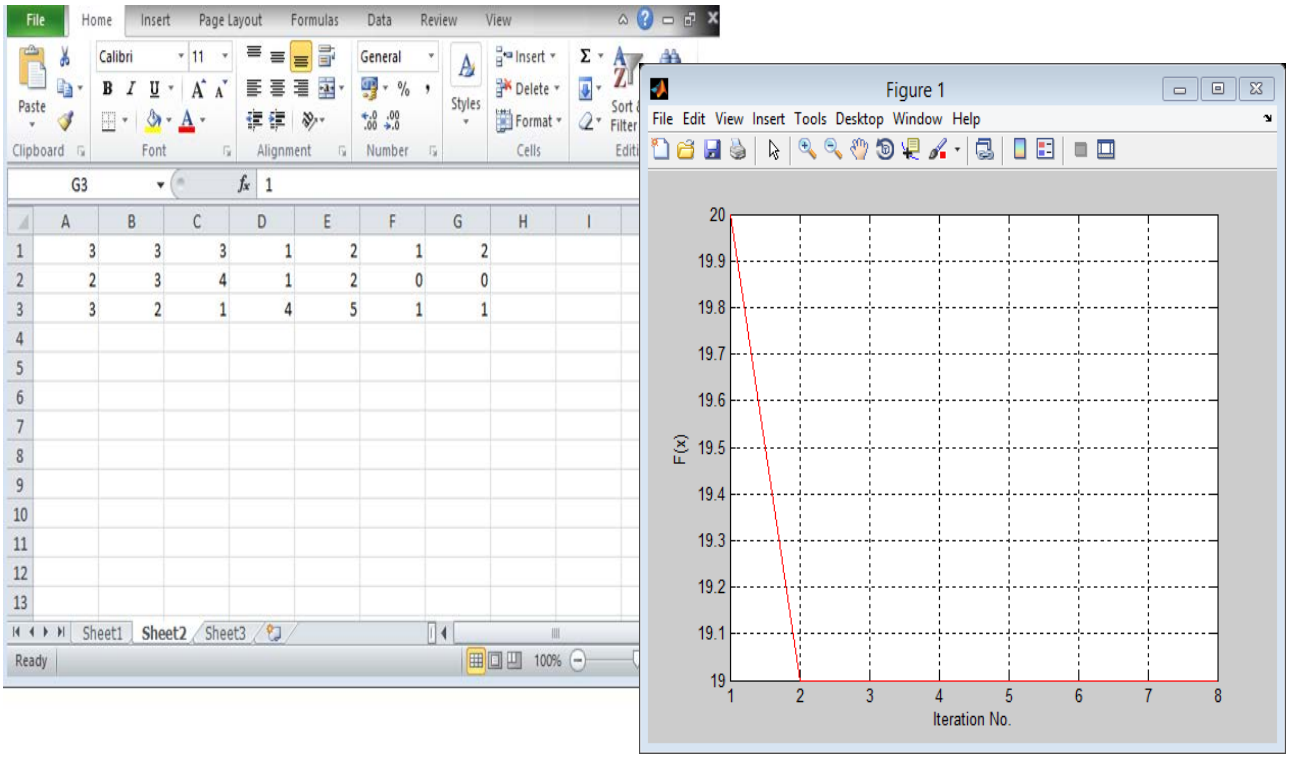

Fig: 6 The corresponding Convergence Graph

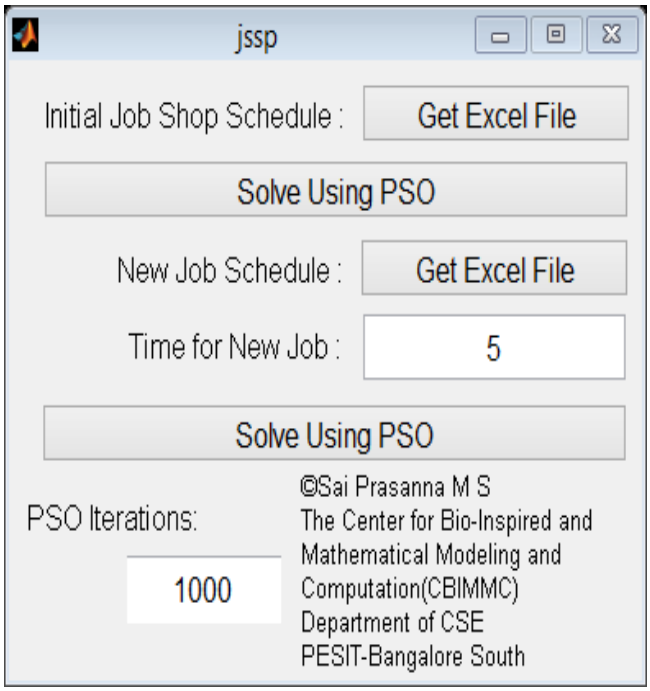

Fig 7 : Arrival of now job at t=5

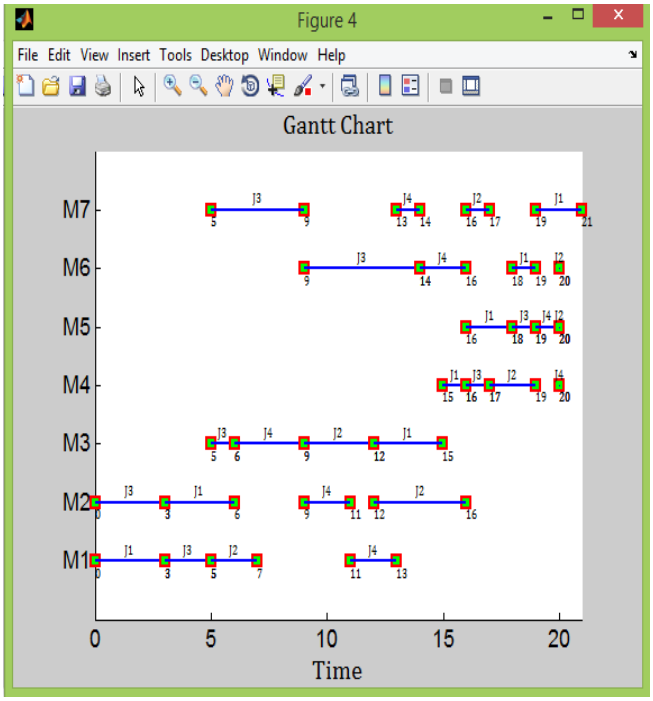

Fig 8: Gantt Chart of the sequence after new arrival

Iteration No.	Best f(x)	Mean f(x)
1	20	23.58
2	19	22.24
3	19	21.86
4	19	21.4
5	19	21.16
6	19	20.96
7	19	20.74
8	19	20.52

PSO Terminated: Convergence Achieved

Fig 9: Output accommodating the New Job

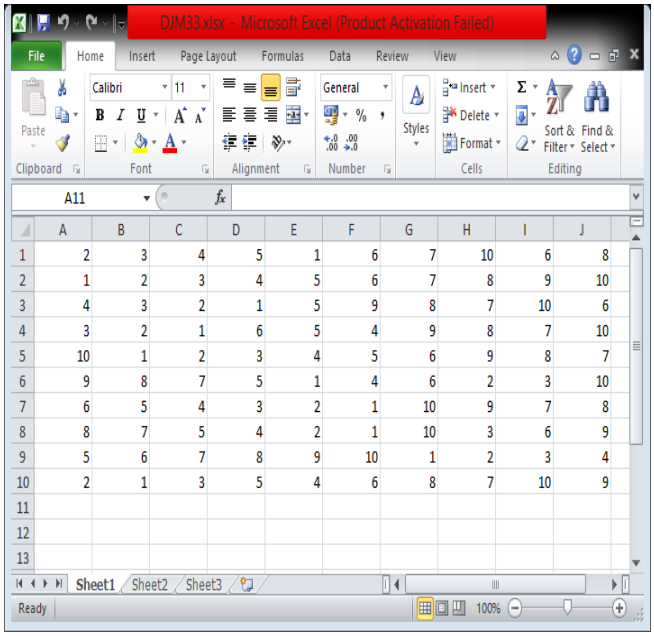

Fig 10: The Problem of size 10*10

Case-2

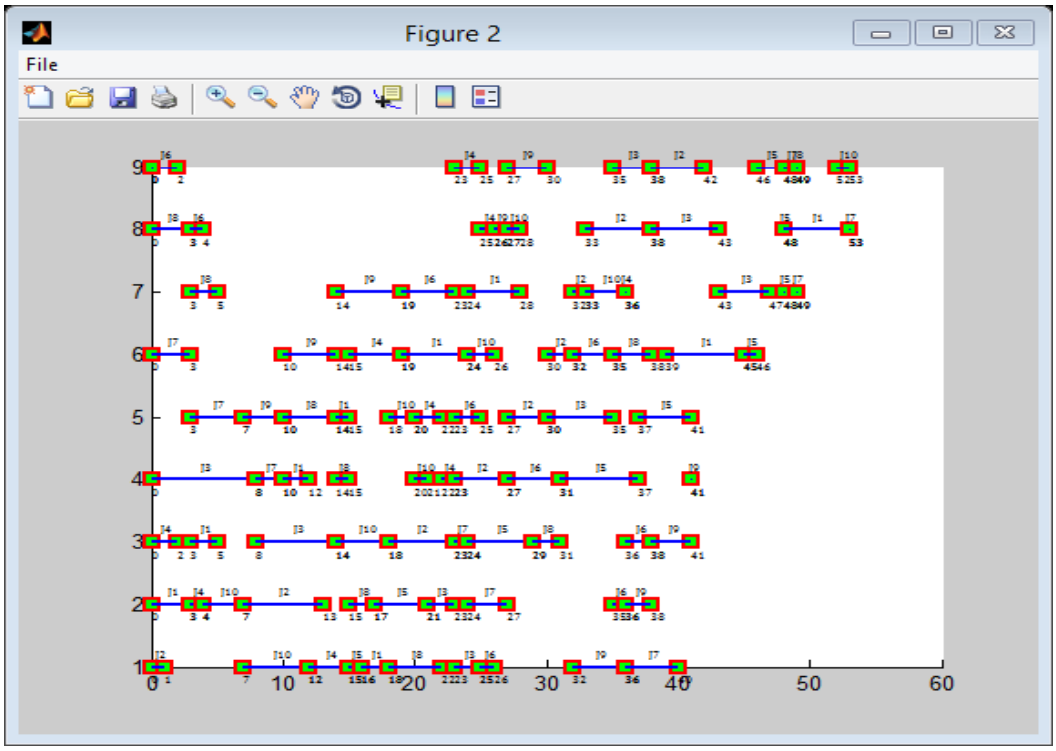

Fig 11: Corresponding Schedule

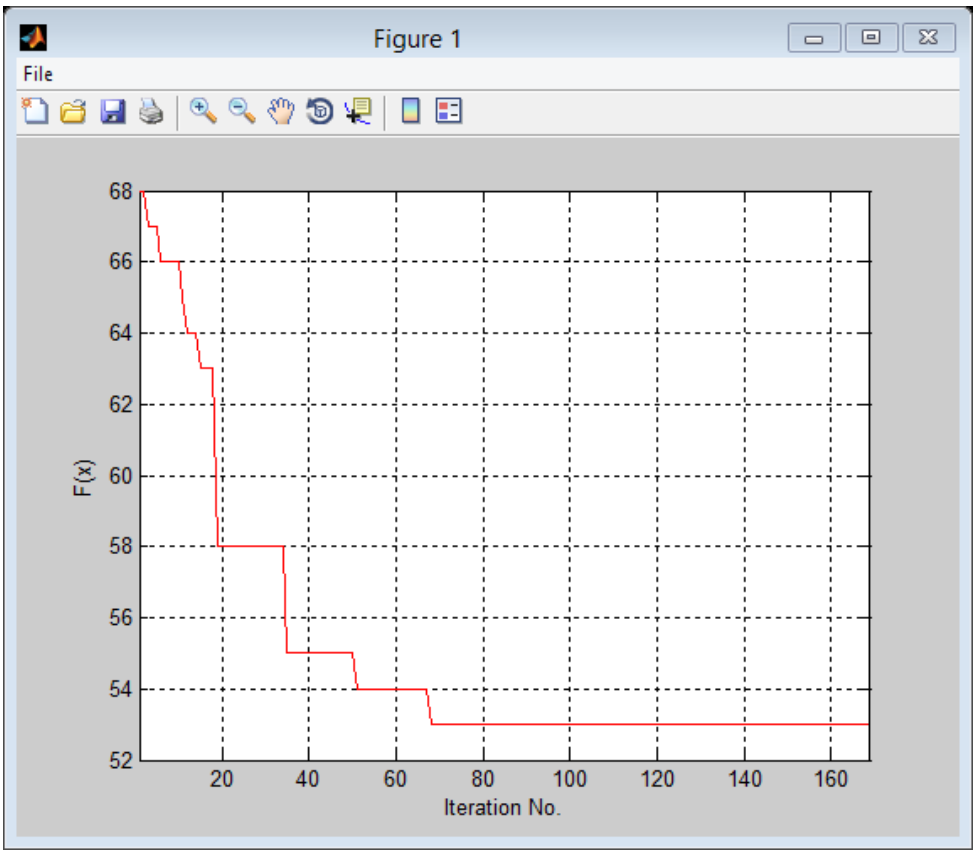

Fig 12: Convergence Graph for 10*10 problem

The above results and Gantt-graphs show the output obtained i.e. the sequence of jobs that yield the best completion time. It was observed that even on a system with average configuration, the time taken to run the heuristics was very less. Currently, this deployment is able to handle an input matrix of size 10*10. However, scalability is very easy to accomplish by a revision of the factors – The maximum iterations possible and the Swarm population size at the beginning of the process as mentioned in the algorithm.

V. CONCLUSION

This paper addresses the DJSSP, which is a significant problem in the real manufacturing scenario. The proposed method uses very minimal computational time to produce an optimal solution. This concept has been implemented using a user friendly interface on a pre-defined dataset and the results obtained meet expectations.

The method is proven to be scalable and performs as expected. This method can be further enhanced to make it more usable in the real environment by considering several additional factors such as the machine shifts, calendars and machine breakdown scenario. A set of self-sufficient agents which keep a check on machine status may be able to achieve the goal.

VI. REFERENCES

1. Kennedy, J.; Eberhart, R. (1995). "Particle Swarm Optimization". *Proceedings of IEEE International Conference on Neural Networks* **IV**. pp. 1942–1948. doi:10.1109/ICNN.1995.488968

2. http://cs.armstrong.edu/saad/csci8100/pso_tutorial.pdf

3. http://ar.newsmth.net/att/22f4075b16b1d1/An_Analysis_of_Particle_Swarm_Optimizers.pdf

4. An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem

   Guohui Zhang, Xinyu Shao, Peigen Li, Liang Gao,Computers & Industrial Engineering 56 (2009) 1309–1318

5. An ant colony system for permutation & flow-shop sequencing Kuo-Ching Yinga, Ching-Jong Liaoa, Computers & Operations Research 31 (2004) 791–801, Elsevier

6. C. Li and S. Yang. A generalized approach to construct benchmark problems for dynamic optimization. Proceedings of the *7th Int. Conf. on Simulated Evolution and Learning*, pp. 391-400, 2008. Springer.

7. C. Li and S. Yang. An adaptive learning particle swarm optimizer for function optimization. Proceedings of the *2009 IEEE Congress on Evolutionary Computation*, pp. 381-388, 2009. IEEE Press.

8. Branke, T. Kaußler, C. Schmidth, and H. Schmeck, "A multi-population approach to dynamic optimization problems" in *Proc. 4th Int. Conf. Adaptive Computing in Design and Manufacturing*, 2000, pp. 299-308.

9. J. Branke, *Evolutionary Optimization in Dynamic Environments*,

10. Kluwer Academic Publishers, 2002.
C. Wei, Z. He, Y. Zhang and W. Pei, "Swarm directions embedded in fast evolutionary programming," in *Proc. Congr. Evolomput.*,vol. 2,2002.