SCHEDULING IN FLEXIBLE MANUFACTURING SYSTEMS (FMS)

S.Kaveri[1], B.Satish Kumar[2], V.Thirupathi[3]
Email:[1]sunkara.kaveri@gmail.com, [2]satishbk91@gmail.com, [3]thirulu629@gmail.com

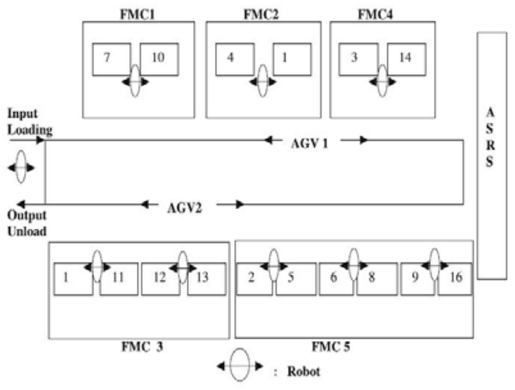
Abstract— Scheduling of jobs in flexible manufacturing systems involves a complicated and complex evaluation of parameters in order to obtain an efficient sequence of processes to be carried on to ensure a continuous production. This is a time taking and tedious task as the set of parameters to be evaluated may range to a large number. The objective of this paper is to develop an algorithm and therefore coded in a software language with all the parameters considered. Necessary conditions are fed as in input to the program which then generates the optimized sequence as an output to be followed by executing the algorithm.

Key Words — FMS, Jobs, scheduling, Machines.

I. INTRODUCTION

Flexible manufacturing systems (FMS) are a group of machines most preferably CNC which are coordinated by a common control centre which has the ability to deal with the variety of products. It is a manufacturing system which possesses the flexibility of adopting its machines and factory environments according to the product to be produced. Though the flexibility in the manufacturing systems is an advantage it proves to be a very complicated task when it comes to scheduling of parts in a changing environment. It has to be done keeping in view of the number of parts to be processed, the number of work stations and the number of individual machines in each work station, customer requirements and many other auxiliary set of variables are to be considered [1].

The objective of this paper is to develop an algorithm which can be applied in solving the problem of scheduling on a large scale with relative ease. Traditional scheduling method does not keep up with the growing requirements of the customers and enterprises to produce the parts with low lead time low cost and instant procurement. To achieve this, an effective algorithm must be used such that an efficient processing sequence is obtained assuring load balance on the work station is maintained in order to avoid heavy or under loading conditions and get use of the machines to a full throughput. It is further made Easy when the algorithm is coded into a software language which enables the operator perform scheduling task with relative ease by simply entering the job parameters and machining parameters as input and when the program is run with as guided by the algorithm the output will be a sequence to be followed in order to avoid uninterrupted production which is unfavorable to any enterprise[2].



Structure of FMS

II. ALGORITHM

When a new job enters the system, it must declare the maximum number of machines of each work station that it may need for its processing. This number must not exceed the total number of machines in the system. When a job requests a set of machines, the system must determine whether the allocation of these machines to that job will leave the system in a safe state. If it will, they are allocated; otherwise, the job must wait until some other jobs get processed.

Data structures are maintained to govern the above activities to store and retrieve the functions when required. Some data structures we use in our algorithm:

Available: A vector of length m indicates the number of available machines of each cell. If available[j]equals k, there are k instances of machines available in station available.

Max: A n*m matrix defines the maximum demand of each job. If Max[i][j] equals k, then job Ji may request at most k instances of station.

Allocation: An n*m matrix defines the number of jobs of each type currently allocated to each machine. If allocation[i][j] equals k, then job is currently allocated k instances of station.

Need: An n*m matrix indicates the remaining machines need of each job. If Need [i][j] equals k, then job may need k more instances of station to complete its task[7].

III. MATH

Safety Algorithm: The algorithm for finding out whether or not a system is in a safe state. This algorithm can be described as follows.

Work and finish be vectors of length m and n respectively. Initialize

Work=Available and Finish[i] =false for i=0, 1,….n-1.

Find an i such that both

a. Finish[i]==false

b. Need =work

If no such I exist, go to step4.

Work= work+Allocation

Finish[i] =true

Go to step2

If Finish[i]==true for all I, then the system is in a safe state.

Machine-Request Algorithm: The algorithm which determine if requests can be safely granted.

Request, be the request vector for job Ji, if Request[j]==k, then job Ji wants k instances of machines type Mj, when a request for machines is made by job Ji, the following actions are taken.

If Request≤ Need, go to step2. Otherwise, raise an error condition, since the job has exceeded its maximum claim.

If Request≤Available, go to step3. Otherwise Ji, must wait, since the machines are not available.

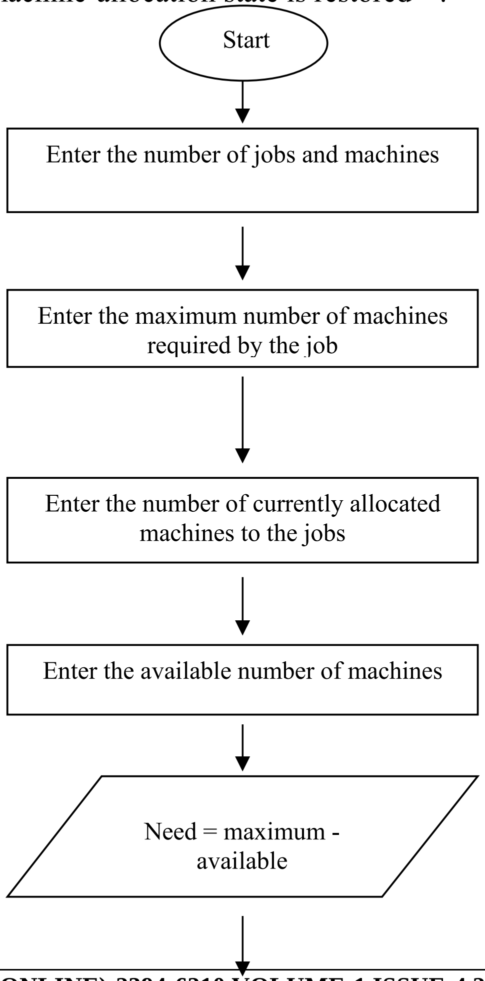
Have the system pretend to have allocated the requested machines to job Ji by modifying the state as follows;

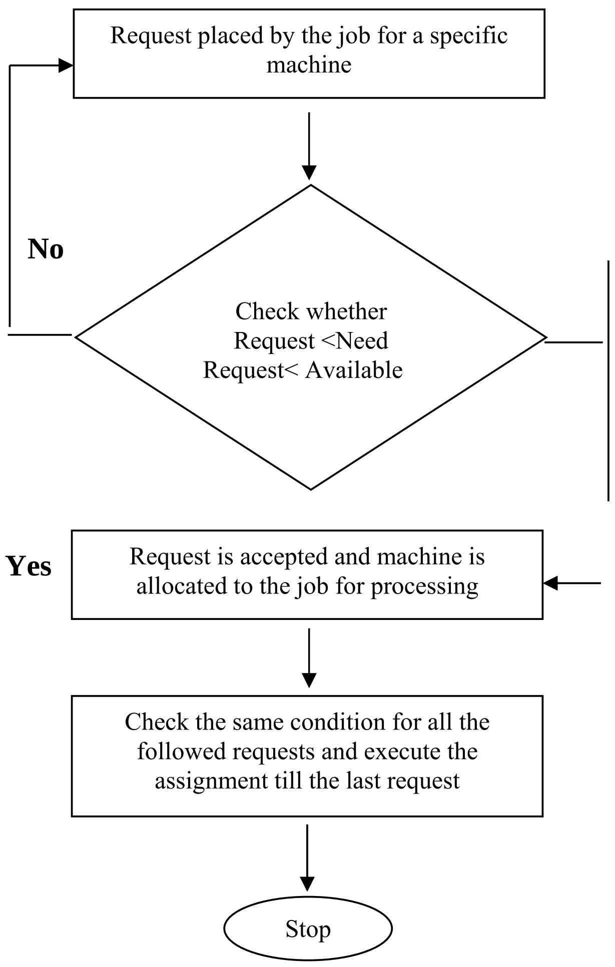
*Available=Available-Request;

*Allocation=Allocation +Request;

*Need=Need-Request;

If the resulting machine-allocation state is safe, the transaction is completed, and Job Ji is allocated its machines. However, if the new state is unsafe, then Ji must wait for Request, and the old machine-allocation state is restored [5].


Start
Enter the number of jobs and machines
Enter the maximum number of machines required by the job
Enter the number of currently allocated machines to the jobs
Enter the available number of machines
Need = maximum - available


Request placed by the job for a specific machine
No
Check whether Request <Need Request< Available
Yes
Request is accepted and machine is allocated to the job for processing
Check the same condition for all the followed requests and execute the assignment till the last request
Stop

EQUATIONS

Available: A vector of length m indicates the number of available machines of each type.

Avail[i][j]=k

Maximum: An n*m matrix defines the maximum number of machines may needed by the jobs for its processing

Max[i][j]=k

Allocation: An n*m matrix defines the number of machines of each type currently allocated to each job.

Alloc[i][j]=k

Request: An n*m matrix indicates the current request of each job if Request[i][j] equals k, then job Ji is requesting k more instances of machine type Mj.

Work and Finish be vectors of length m and n respectively, initialize Work=Available. For i=0,1…n-1. If allocation≠0, then finish[i]=false otherwise finish[i]=true

Find an index I such that both

a. Finish[i]==false

b. Request≤work

If no such I exist, go to step4

Work=work+allocation

Finish[i]=true

Go to step2

If finish[i]==false for somei,0≤i<n, then the system is in a deadlock state.

Moreover if finish[i]==false then job Ji is deadlocked.

IV. RESULTS

The necessary parameters when given to the program as input which includes the set containing the total number of jobs, work stations and the machines in each work station, status of maximum, currently allocated, available machines on the work floor and other miscellaneous factors like processing time limit, instances of each machine when are executed the output obtained is a sequence of jobs to processed so as to avoid any kind of deadlock and breakdown on the shop floor. Therefore for effective results and optimized usage of machines on the shop floor the scheduling has to be done according to the sequence obtained by as the output by the program.

If at all there is no proper sequence which can be generated by the program for the given set of values then it generates an incomplete sequence of jobs to be processed up to which no blocking or starving conditions prevail in the system leaving the system in a safe operating state. It notifies that the processing of immediate next job after this sequence may lead to a blocking or starving leaving the system in an unsafe state. Additionally an avoidance algorithm can be used to solve these kinds of situations [7].

V. CONCLUSION

Optimization procedure has been developed in this work which is based on banker's algorithm and is implemented successfully for solving the scheduling optimization problem of FMS. Codes are written in C++ language. Results are obtained for n jobs and m machines in the system. With less computational effort it is possible to obtain the solution for such a large number of jobs and machines .This work leads to the conclusion that the procedures developed in this work can be suitably modified to any kind of FMS with a large number of components and machines subject to multi objective functions.

If research carried on the algorithm and re writing of the program in other advanced software languages its application can be implemented in a variety of fields which will include availability and handling times of loading/unloading stations, robots and AGVs.

VI. FUTURE SCOPE

- Multi objective scheduling for FMS can be done by further development of the algorithm unlike only job shop scheduling done here.
- A deadlock prevention phenomenon is applied in this paper which can be more sophisticated by applying deadlock avoidance strategy.
- The algorithm written here is coded in C++ language which when coded in a further sophisticated software language the capability, accessibility and responsive speed can be enhanced.
- The coded language when interfaced with animation software like FLEXSIM and AUTOMOD the visual experience can also provided.

REFERENCES

[1] Fang, H.-L., Ross, P., and Corne, D. (1993). A promising genetic algorithm approach to job-shop

[2] scheduling, rescheduling, and open-shop scheduling problems. In Forrest, S., editor, Proceedings of the Fifth International Conference on Genetic Algorithms, pages 375–382, Morgan Kaufmann, SanMateo, California.

[3] International Journal of Production Research ISSN 0020±7543 print/ISSN 1366±588X online # 2002 Taylor & Francis Ltd

[4] Ram Pratap Yadav *et al* / VSRD International Journal of Mechanical, Auto. & Prod. Engg. Vol. 2 (7), 2012

[5] Lin, S., Goodman, E., and Punch, W. (1997). A genetic algorithm approach to dynamic job shop scheduling problems. In B¨ack, T., editor, Proceedings of the Seventh International Conference on Genetic Algorithms, pages 481–489, Morgan Kaufmann, San Mateo, California.

[6] Spyros A. Reveliotis. Conflict Resolution in AGV Systems. School of Industrial & Systems Engineering, Georgia Institute of Technology.

[7] Benjamin Zhan F. (1996). Three Fastest Shortest Path Algorithms on Real Road Networks: Data Structures and Procedures. Journal of Geographic Information and Decision Analysis, vol.1, no.1, pp. 69-82.

[8] Chang, W.K., Tanchoco, J.M.A., and Koo, P.H. (2005). Deadlock Prevention in Manufacturing Systems with AGV Systems: Banker's Algorithm Approach. Journal of Manufacturing Science and Engineering,v119, 849-854.

[9] Hyuenbo, Cho, Kumaran, T.K., and Richard, A.Wysk. (1995). Graph Theoretic Deadlock Detection and Resolution for Flexible Manufacturing Systems. IEEE Transactions on Robotics and Automation, v11, n3,413-421.

[10] Stefano Pallottino, Maria Grazia Scutella (1998). Shortest path algorithms in transportation models: Classical and innovative aspects. University of Pisa.

[11] ARAKI, T., TAKAHASHI, T., SUEKANE, M., & KAWAI, M. Flexible AGV system simulator Proceedings of the 5th International Conference on AGV Systems, 77-86.Ram Pratap Yadav et al / VSRD International Journal of Mechanical, Auto. & Prod. Engg. Vol. 2 (7), 2012282

[12] DEWSNUP, M. C. (1995) How to model AGVS using ProModel for Windows. Proceedings of 1995 Winter Simulation Conf., 482-486.

[13]Yeh, M.S., and Yeh, W.C. (1998). Deadlock Prediction and avoidance for zonecontrol AGVS.INT.J.PROD.RES, v36, n10, 2879-2889.

[14]Storer, R.,Wu, S., and Vaccari, R. (1992). New search spaces for sequencing problems with application to job shop scheduling. Management Science, 38:1495–1509.

[15] Taillard, E. (1993). Benchmarks for basic scheduling problems. European Journal of Operational Research, 64:287–295.