Numerical Simulation and Assessment of Meta Heuristic Optimization Based Multi Objective Dynamic Job Shop Scheduling System

Bhumika Singh Rajput*, Anshul Gangele*, Satish Kumar Alaria*

a Student, Department of Computer Science &Engineering, Medi Caps University, Indore, India
b Professor, Department of Mechanical Engineering, Adina Institute of Science and Technology, Sagar, India,
c Assistant Professor, Department of Computer Science & Engineering, AIET, Jaipur

Abstract—In today’s world of manufacturing, cost reduction becomes one of the most important issues. A successful business needs to reduce its cost to be competitive. The programming of the machine is playing an important role in production planning and control as a tool to help manufacturers reduce their costs maximizing the use of their resources. The programming problem is not only limited to the programming of the machine, but also covers many other areas such as computer and information technology and communication. From the definition, programming is an art that involves allocating, planning the allocation and utilization of resources to achieve a goal. The aim of the program is complete tasks in a reasonable amount of time. This reasonableness is a performance measure of how well the resources are allocated to tasks. Time or time-dependent functions are always it used as performance measures. The objectives of this research are to develop Intelligent Search Heuristic algorithms (ISHA) for equal and variable size sub lot for m machine flow shop problems, to Implement Particle Swarm Optimization algorithm (PSO) in matlab, to develop PSO based Optimization program for efficient job shop scheduling problem. The work also address solution to observe and verify results of PSO based Job Shop Scheduling with help of graph chart.

Index Terms—Job Shop, PSO, Optimization, Dynamic Work Span, Scheduling

I. INTRODUCTION

Cost cutting has emerged as one of the most crucial concerns in the modern manufacturing industry. A profitable company must lower its costs to remain competitive. The machine's programming is a crucial component of production planning and control since it enables manufacturers to cut costs while making the best possible use of their resources. The programming issue is not just restricted to machine programming; it also affects a wide range of other fields, including computer, information, and communication technologies. According to the definition, programming is an art that entails allocating, organizing, and utilizing resources in order to accomplish a goal. The program's goal is to finish tasks within a fair period of time. A performance indicator of how well resources are allocated to tasks is reasonableness. It always uses time or time-dependent functions as performance indicators.

The majority of research on genetic programming algorithm optimization uses an evolutionary and genetic "living of the strongest "rule. According to Jain and Meera [11], this strategy has been effective for evolutionary computation using JSP, but it hasn't been demonstrated to be better than other machine intelligence methods.

A form of computational intelligence known as particle swarm optimization has demonstrated promise for resolving additional optimization issues (PSO). The flight patterns of a flock of birds served as inspiration for the PSO Algorithm. It is regarded by Song and Gu [19] as one of the quicker convergent computational algorithms for intelligence and a prime candidate as an algorithm for multimodal functions. Since the PSO optimization approach has been used to solve other scheduling issues, such as the flow line problem (FSP) and the versatile work shop shop problem (FJSP), which are related issues that will be discussed later, see [12], [21], [22], and [25]. One benefit of the PSO algorithm is its speed. It is praised as a very quick convergence algorithm. The problem is only suitable to try to make this relation using the PSO algorithm JSP given the performance of strategy optimization in other areas and the ongoing need for improved heuristic methods to solve Workshop problems. Since creating a feasible schedule can
be very computationally expensive. Therefore, any method that can prove its value would be more beneficial.

![Figure 1. Job-Shop Problem](image)

Of course, each new method should be at least briefly explored to determine whether it may eventually result in more intriguing research and, in turn, better optimization techniques. One strategy has been used to make this optimization achievable because of the continuous nature of the PSO and a JSP permutation.

II. JOB SHOP SCHEDULING

A (n m) Shop job problem has a set of jobs, n, each with an order of operations, m, that is equal to the number of machines or resources specified in the problem. As a result, a work Ji is a predetermined sequence of operations: Oi (Oi, 1, Oi, 2, ..., Oi, m). Oh, no. Oij, each service has a running period, also known as working hours. The following are the rules for traditional JSP:

- Each job can only be handled by one computer at a time.
- Each machine can only do one job at a time.
- With the use of m computers, all jobs must be processed in the same order.
- At time zero, all jobs and machines are operational.
- Following the completion of the previous operation, a new operation begins.
- Each job’s processing and setup times are known.
- Each work is self-contained, and there are no restrictions on priority.
- Machine breakdowns and interruptions are not taken into account when processing.

- Computers may not be idle during the execution of a task, but they may be idle during the transition between jobs.
- Sublot sizes should be understood ahead of time and remain consistent.
- It is not permitted to interrupt the processing of a sublot.
- Setup time for sub lots belonging to different batches can be needed.

It's easier to imagine what a semi-active programme isn't to comprehend this idea.

<table>
<thead>
<tr>
<th>Job</th>
<th>Machine Sequence (Processing Time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td>1 (3)</td>
</tr>
<tr>
<td>Job 2</td>
<td>1 (5)</td>
</tr>
<tr>
<td>Job 3</td>
<td>2 (4)</td>
</tr>
</tbody>
</table>

Figure 3.2 shows a Gantt chart depicting one non-semi-active schedule for this simple (3×3) JSP.

![Figure 3.2. Gantt chart of schedule for (3×3) JSP](image)

Keep in mind that the second and third procedures on machine 1 were not scheduled for the first available time window. The red boxes indicate this unnecessary extended wait. The times are obviously not ideal. Simply make the second calendar third actions on machine 1 semi-active by moving from left to the earliest permissible time. In Figure 3.2, this new semi-active schedule is displayed. It's important to note that none of the aforementioned activities may be begun without changing the order in which the machine performs its actions. [27]

A direct depiction of a schedule is any design that explicitly states the time when each operation will start running on the machines. The scheduling space is directly optimized, potentially via left-shifting. Another example would be to immediately create optimization values based on creating makespan and (N m) set start timings for a (N m) Workshop issues. A meta-heuristic algorithm may be used to learn or develop a set of start timings for this kind of connection based merely on a goal value as the...
makespan. It is evident that when direct rendering is employed, no workable schedules can be generated since the problem's prior constraints are ignored. If start timings for each operation were utilized as a direct representation, the final solution may resemble this if $t_{ij}$ is the operation's start time.

\[
\begin{bmatrix}
8.0 & 0.0 & 5.0 \\
3.0 & 1.0 & 9.0 \\
5.0 & 1.0 & 0.0
\end{bmatrix}
\]

One common method for avoiding precedent constraints in JSP programming is to use an indirect representation and then use a scheduling algorithm to convert the indirect representation into a feasible schedule. Permutation with repetition - In this representation, a scheduling algorithm decodes a permutation of job numbers $(n \times m)$ into a feasible schedule. Each digit represents a job, which is repeated $m$ times. The operation of order $k$ to be processed by job $j$ is represented by the repeated work order $k$. This semi-scheduling algorithm uses active default schedules, such as asset class schedules and no delay. Figure 3.2 shows an example of a permutation with repetition and a scheduler.

![Permutation with Repetition](image)

Figure 3.12: Permutation with repetition

It's simple to see how an indirect representation combined with a corresponding scheduling algorithm can circumvent the limitations of search decision space (the $(n \times m)$ permutation of numbers in the example above). Because, before scheduling, a scheduling algorithm will consult the preceding limitations. The repeat permutation algorithm is a straightforward and efficient programming method.

**III. PSO BASED JOB=SHOP**

The With a number of permutations, the JSP/PSO method was devised to describe an indirect solution to a JSP. There are two obstacles in the way of PSO's immediate use in JSP. The governing equations PSO cannot be used to directly optimize a permutation of the integers that would indicate a solution to the JSP (in the context of a scheduling algorithm), as illustrated in Figure 4.1. There must be a technique to continuously permute space. The choice of what kind of scheduling algorithm to utilize, or how to turn these permutations of integers one at a time, is the second roadblock. Figure 5.1 on the following page of the complete clarifying process illustrates these two independent stages.

The permutation decoding process is shown in green space, while the transformation process is shown in blue space. The continuum of particles in the space permutation were altered using the greatest value prior rule, or GVP [15]. (shown in blue in Figure 5.1). The GVP rule includes giving each dimension or component of a particle continuous space its complete index. The sequence in which these distributions are formed determines their permutation. The $n$ permutation values will be high if there is an $n$-dimensional continuous PSO space. The permutation of a particle is determined by assigning a value of 1 to the dimension with the largest magnitude. Then, a value of two is assigned to the particle's dimension with the next-largest magnitude. For every aspect of the issue, this procedure is repeated. The foregoing method converts a particle's three-dimensional space into a permutation, as shown in Figure 5.2. [22]
Always be combined to build a solution because

the world if this occurs and

position reached by the particle in

considers a better makespan. Each swarm will record the

distinct swarms, when a collection of connected particles

There are m better separated places, one for each of the m

division principle does not initially appear to be effective

cle

m

5.6.

example is to show how space is divided. Anothe

locations of particles in space, however the goal of this

the permutation as permutations are created from the

permutation as permutations are created from the

of a particular shows it graphicall

have the same “fitness” throughout each cycle. Figure 5.8 in

swarm will a

be chosen at random. The identical particles from each

optimization. The pairing will soon be established and will

There are particles, however the swarm size is

sense because the priority list for each machine must differ.

There are m better separated places, one for each of the m

different swarms, when a collection of connected particles

consider a better makespan. Each swarm will record the

position reached by the particle in the swarm as the best in

The arrows technically should point from the particle to

the permutation as permutations are created from the

locations of particles in space, however the goal of this

e xample is to show how space is divided. Another

illustration of the best-of-all principle is shown in Figure

5.6.

Using this technique, the search space in the JSP / PSO has
effectively become m number of swarms, one for each

machine. Figure 5.7 shows this

clearly other "related" particles in other swarms. The space

division principle does not initially appear to be effective

as an optimization tool, and it does not combine well with

other meta-heuristics, but the PSO may be used in this

case because it enables the particles to store information

about their best individual and overall positions in space.

There are m better separated places, one for each of the m

distinct swarms, when a collection of connected particles

considers a better makespan. Each swarm will record the

position reached by the particle in the swarm as the best in

the world if this occurs and makespan is the best to date.

IV. SIMULATION & RESULT

In the MATLAB programming language, the JSP/PSO

was written and evaluated. This algorithm is divided into
two parts: scheduling and optimization, also known as PSO.

Many of the scheduling and optimization (PSO)

aspects have been discussed in general. However, in

the following two pages, more information on how this

software worked in these two areas is revealed.

Each JSP machine has its own swarm, which gives particles

a goal-oriented ideal world to work toward. Which makes

sense because the priority list for each machine must differ.

There are particles, however the swarm size is just 7. In the

JSP/PSO method, each swarm particle is "linked" to another

swarm particle and is maintained as "connected" during

optimization. The pairing will soon be established and will

be chosen at random. The identical particles from each

swarm will always be combined to build a solution because

there is, of course, no genuine real link (which does not

share information). In other words, the "matching" particles

have the same "fitness" throughout each cycle. Figure 5.8 in

particular shows it graphically.

This is a vital idea to grasp. Essentially, these particles

travel independently of other swarms, but their “fitness” or
better location in space is inevitably influenced by the position of

We discovered that using the PSO Optimization process, the job's make time was drastically reduced. With the help of the following table, we have represented the reduction of make span (Objective function). The table below shows a comparison of make span reduction as we perform iterations in the PSO optimization process.

Case-2 - JSP/PSO Results (10 X 10 Problem)

Fig 6.3 PSO based JSP for 10 X 10 Problem (b) Minimization of Objective (Make span)
V. CONCLUSION

The JSP/PSO Algorithm that has been presented can be used to tackle Job Shop Problems. This is significant because of how the optimization process operated, partitioning the search space across the participating machines. This space division approach shows that the existence of such a “collective effort” as of this writing need not be known to individual swarms or communities cooperating towards a shared aim. Although this may have been acknowledged in the meta-heuristic optimization community, to the best of my knowledge, it has never been used to solve the Work Shop Problem. A more important finding is that an algorithm may have a greater likelihood of success if it employs the same search space split by machines but promotes information exchange amongst various swarms or populations.

REFERENCES

[6]. Fenton, P., Walsh, P., Improving the Performance of the Repeating Permutation Representation Using


