Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

ITERATIVE SIMULATION AND OPTIMIZATION APPROACH FOR JOB SHOP SCHEDULING

Ketki Kulkarni
Jayendran Venkateswaran

Industrial Engineering and Operations Research
Indian Institute of Technology Bombay
Room 308A, Mechanical Engineering Building
Mumbai, MH 400076, INDIA

ABSTRACT

In this paper, we present an iterative scheme integrating simulation with an optimization model, for solving complex problems, viz., job shop scheduling. The classical job shop scheduling problem which is NP-Hard, has often been modelled as Mixed-Integer Programming (MIP) model and solved using exact algorithms (for example, branch-and-bound and branch-and-cut) or using meta-heuristics (for example, Genetic Algorithm, Particle Swarm Optimization and Simulated Annealing). In the proposed Iterative Simulation-Optimization (ISO) approach, we use a modified formulation of the scheduling problem where the operational aspects of the job shop are captured only in the simulation model. Two new decision variables, *controller delays* and *queue priorities* are used to introduce feedback constraints, that help exchange information between the two models. The proposed method is tested using benchmark instances from the OR library. The results indicate that the method gives near optimal schedules in a reasonable computational time.

1 INTRODUCTION

Simulation is used as an evaluation tool in many decision support systems. The framework of simulation-based optimization, where a discrete-event simulation model is interfaced with an optimum seeking package, is a popular methodology used for complex systems. Here, a discrete-event simulation model is integrated with a meta-heuristic based solver. The simulation model evaluates the makespan for a given combination of inputs suggested by the solver. This process goes on iteratively, for a specified number of iterations.

For large and complex systems with numerous interacting elements, simulation models alongside analytic methods are used so that large number of alternatives are compared without damaging the actual system or incurring huge costs. Simulation has an advantage over analytic techniques because it does not require extensive use of simplifying assumptions, allowing it to better model the uncertainty in complex systems (Glowacka, Henry, and May 2009). Simulation has been integrated with various analytic techniques such as linear programming (Bang and Kim 2010), heuristic methods (Jeong, Lim, and Kim 2006), Data Envelopment Analysis (McMullen and Frazier 1999). These hybrid models cover a large range of applications including manufacturing systems, assembly lines, health care and supply chains. For some applications, fast analytic models do exist. However, solutions are not always applicable to real systems due to randomness that is not modelled in the analytic methods. Extensive use of these models is found in scheduling and planning of manufacturing systems to attain feasible and realizable solutions.

A class of hybrid models using Mathematical Programming Representations (MPRs) instead of detailed simulation models is presented by Schruben (2000). Schruben (2000) presents analytic methods to capture the event-based dynamics of queuing systems, which can be an alternative to simulation models for queuing systems. In this approach, discrete event simulation models are represented as event graphs. The state trajectories of the event graphs are solutions to mixed integer programs or linear programs. Analytic models specific to the problem are developed for the event graph that captures the system dynamics. These analytic models are shown to be solved quickly, as compared to conventional simulation methodology. Matta (2008) extends the work by Schruben (2000) and presents an application of this hybrid model for flow lines with buffers. An exact mixed integer linear model is compared with the proposed approximate LP model and a stochastic programming model. The LP-approximate model is claimed to be fast and near-optimal.

Byrne and Bakir (1999) present a framework which combines analytic and simulation modelling methods in an iterative hybrid approach. The analytical model is a linear programming(LP) formulation, while a simulation model accommodated manufacturing system characteristics, such as queuing and transportation delays. In this planning problem, the LP model runs first with deterministic inputs. Since the simulation models the shop floor in detail, the realizability of the suggested 'optimal' plan is evaluated. The feedback from the simulation is used to update the capacity constraint in the LP model. This loop is run iteratively till some specified stopping criteria is achieved. Here, the mathematical program generates a solution that is optimal, but not necessarily in the real system. The simulation model evaluates the realizability of a given solution. Therefore, the system moves towards a realizable solution that need not be globally optimal.

Lee and Kim (2000) extend the work of Byrne and Bakir (1999) and present a hybrid method combining the analytic and simulation model for an integrated production-distribution system in supply chain environment. The solution procedure of independently developed analytic and simulation model were used together to solve the problem. The focus was to obtain a realizable solution, which need not be globally optimal.

Bang and Kim (2010) proposed a Hierarchical Production Planning(HPP) method in a semiconductor wafer fabrication facility. An LP model is used for production planning at the higher level. At the lower level, a list scheduling method is used for detailed scheduling and discrete-event simulation is used for feasibility/desirability check of the production plan. An iterative method was proposed, in which waiting times are updated and product types are regrouped for the next iteration according to the simulation results of the current iteration, if the production plan obtained from the LP model is found to be undesirable during the simulation run.

Mahdavi, Shirazi, and Solimanpur (2010) developed a simulation-based decision support system for a stochastic flexible job shop. Here, a simulation model evaluates real time inputs from the shop floor. A rule-based engine compares the output from the simulation model with the expected output. An analytic model based on the Response Surface Methodology (RSM) computes the necessary changes to the system in order to obtain the expected output. The proposed changes are run on the simulation model and this process continues iteratively till the output of the simulation model matches the expected output. Once again the focus being to obtain a realizable solution.

We reviewed several papers that used iterative schemes involving simulation and an analytic model. An interesting factor is the number of times the simulation model interacts with the analytic model. The system may be open loop (run once) or closed loop (iterative interactions). NP-hard problems like scheduling are good candidates for iterative methods, where suitable feedback at each iteration brings the analytic model closer to the optimum. Simulation when used at design level could be open loop, where multiple alternatives are compared and evaluated. Open loop may have multiple runs of the simulation model. However, the interaction between the simulation and analytic model is not iterative.

In this paper, we present an iterative modelling approach using a discrete-event simulation model and an optimization routine to address the job shop scheduling problem. This architecture is based on the hierarchical production planning problems framework implemented by Byrne and Bakir (1999). However, we extend this approach to obtain an optimum solution. In the scheduling problem, the use of simulation model ensures that any output from simulation is a feasible solution. Hence, the challenge is to move towards optimality, given the quality of the solution. In the proposed method, a Linear Programming (LP) model helps move towards optimality, after receiving inputs from a feasible solution given by the simulation model. Identifying the information to be exchanged between the two models (LP and simulation) and the mechanism to update the LP so as to move towards optimum are challenging tasks.

1.1 Job Shop Scheduling Problem (JSSP)

The JSSP is a well known NP-hard problem. While exact methods exist to solve small and medium sized deterministic problems (Pan and Chen 2003), the performance of these methods worsens drastically for large sized problems. There are many variants of the job shop problem such as classical and flexible job shop scheduling. In general, the objective of scheduling problems is to assign and sequence p operations of n jobs on m machines so as to minimize makespan or maximal completion time. Attaining a global optimal solution is NP-Hard, even without stochasticity. With hybrid models, that is, models using more than one type of sub-models interacting with each other, we aim to find near optimal solutions within reasonable computational time.

Apart from achieving optimum value of objective function, the solution may need to satisfy certain criteria such as utilization of each machine above/below certain threshold, Time Between Departures (TBD) below threshold value and number of tasks scheduled per machine below certain threshold (Mahdavi, Shirazi, and Solimanpur 2010). These are often collectively called bottleneck resolution criteria and thresholds depend on desired system performance.

The classical job shop scheduling problem (CJSSP) has a fixed input sequence for each job, where every job has to visit every machine once and only once. The objective is to find a schedule for each machine (order in which to process jobs) that minimizes the total completion time for the system (makespan). Pan and Chen (2003) have presented an exhaustive literature survey of exact methods used to address the JSSP. It is concluded that the Mixed Integer Programming (MIP) model proposed by Manne (1959) gives the best performance for this class of problems.

Apart from exact methods, another popular solution approach for this class of problems is the use of meta-heuristics. Different algorithms such as Genetic Algorithm, Simulated Annealing and Tabu Search have been used (van Laarhoven et al. 1992; Klemmt et al. 2009). In general meta-heuristics use some form of randomization, to explore the solution space and accept a sub-optimal solution with some probability. The randomization helps them to escape local optima and makes it possible to move sooner towards a better solution. The meta-heuristics treat the system like a black-box, making no assumption about its behaviour.

We propose a method that can escape local optima, similar to meta-heuristic based approaches. However, it does not treat the system like a black-box. System specific knowledge can be included in the model in the form of constraints, without worsening the performance. This is in contrast to the simulation-based optimization models, where adding additional constraints can sharply increase the time required to obtain even a feasible solution. The proposed method uses feedback from previous runs to move across the solution space, as opposed to use of randomness by meta-heuristics. Due to this, the experiments are reproducible and the solution remains unchanged, each time the model is run. The meta-heuristics need not guarantee the same outcome for each run of the model.

In the following sections, we present a formulation for CJSSP based on the hybrid architecture described so far. The iterative scheme is explained by presenting it in the form of an algorithm. The variables involved, flow of information, stopping criteria and constraints are also explained in detail. The final algorithm is referred to as 'Iterative Simulation Optimization (ISO)'. While the scheme yields itself naturally to model stochastic problems (due to the use of simulation model), the concept is demonstrated in this paper using deterministic benchmark instances from the OR library (OR-library 2014).

2 SIMULATION MODEL

The operations of the job shop are modelled using discrete-event simulation, with each machine modelled as a unit capacity resource with an infinite capacity queue in front of it. Every job is an entity of the model, having a pre-defined sequence of machines and the process time on each machine associated with it. If an arriving job finds a machine busy, it is placed in a queue before the machine. This simulation model is a deterministic model and exactly 1 entity is created for each job.

When this discrete-event simulation model is executed, it is observed that all the entities, that is, jobs are created at time instance 0. The jobs then attempt to 'seize' or gain access to the first machine in their sequence. Ties are broken arbitrarily. The entity which successfully seizes the machine begins processing and the rest are put in a queue before the machine, ordered in first-come order (default). Once the job finishes processing at a machine it proceeds to the next machine as per its sequence. If the job finds the required machine busy, it joins the queue for that machine. Every job proceeds in this manner through the job shop until all the machines in it's sequence are visited. The simulation ends when the last job or entity completes processing at the last machine in it's sequence and the end time indicates the makespan.

As mentioned earlier the execution of the simulation model always yields a feasible schedule even though it may be far from optimum, since the simulation model ensures that any machine can process only 1 job at a given time and the job sequences are strictly followed.

3 THE PROPOSED ISO APPROACH

In this hybrid approach, simulation is iterated with a Linear Programming model (LP). Figure 1 shows the interaction between LP and DES model and the variables exchanged between them. The discrete-event simulation model incorporates the system details governing the resource sharing and sequencing. The schedule for the machines is obtained from the output of the Linear Programming (LP) model. The DES model initially runs with all decision variable values set at zero. Selected data from the simulation run is then passed on to the LP model, which updates corresponding constraints and suggests a new combination of decision variables, that is, schedule.The DES model runs with the new schedule and this process continues iteratively till the specified number of iterations is reached. The best observed schedule is reported in the end, along with the corresponding values of decision variables.

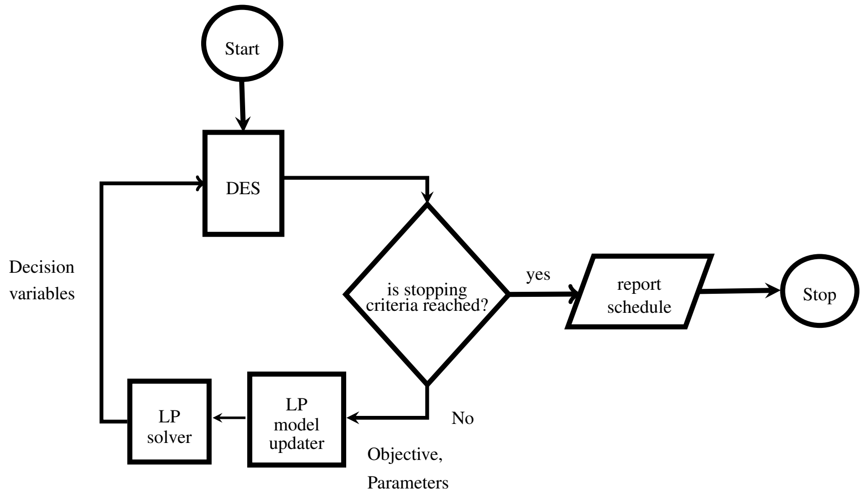


Figure 1: Iterative simulation-optimization.

For a deterministic job shop scheduling problem with fixed processing times, the solution space contains all possible schedules for the machines. The feasible region is determined by the constraints on resources (machines) such as: only one job can be processed on a machine at a time and the process at step $l+1$ cannot begin till the process at step l is complete (sequence constraints). These constraints are modelled using simulation. Therefore, the simulation model defines the feasible set.

Now, in order to find a 'good' schedule that gives a smaller value of the objective function (makespan), we need to intelligently search in the feasible region. For NP-hard problems such as the CJSSP, gradient based searches such as steepest ascent or descent are inefficient because it is hard for these methods to escape the local optima and move towards a different region in the feasible solution space. Meta-heuristics use some randomness to escape local optima and hence have shown better performance for such problems, even though they do not guarantee reaching the optimal solution (due to the inherent randomness in the method).

In the proposed ISO method, while the simulation model ensures feasibility, the optimization model improves the objective function. The constraints the optimization model factor in the feedback from the previous simulation run and further reduce the feasible region to search. During successive runs, different areas of the feasible region are explored each time, determined by the changing constraints in the optimization model. The presence of the simulation model ensures that all solutions are feasible. However, since the constraints in the optimization model change dynamically, it is hard to prove that the optimal solution will eventually be obtained.

The simulation model defines the feasible region and the optimization model provides the search direction for a better solution. A few constraints are added to the optimization model to refine the search by reducing the feasible region. Figure 2 shows the mapping of the solutions from the LP solver to the solutions from the simulation model. The mapping is of 'many-to-one' type, where multiple optimization solutions correspond to the same feasible schedule (simulation solution).

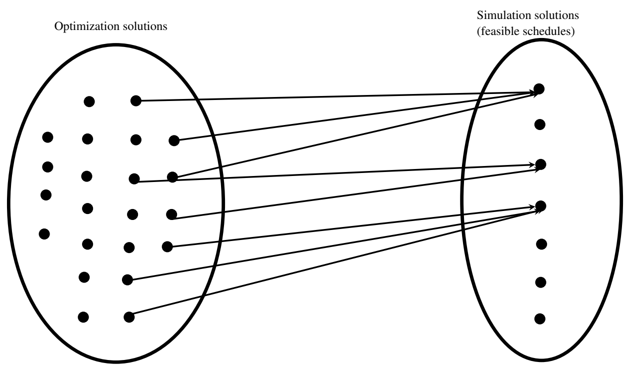


Figure 2: Mapping of solutions.

4 DECISION VARIABLES

The job shop operations are captured by the simulation model. Now, we need to design the decision variables appropriately so as to ensure that the entire sample space of machine schedules is explored. The first natural choice of decision variable are 'queue priorities'.

Queue priorities can be defined as the order of importance of the jobs on each machine. A set of queue priorities is defined for every machine. Of all the jobs waiting in the queue of a machine, the highest priority job is selected for processing first. Therefore, a job arriving later in the queue, but having a high priority, will be processed before a job that was already present in the queue when it arrived.

Once a job finishes processing on a machine, it attempts to 'seize' the next machine in its sequence. If this machine is idle, the job is taken up for processing. This behaviour is the default behaviour of the simulation model and is referred to as 'non-delay' scheduling, indicating no delays in seizing of a resource, once the resource and the job are idle. However, this behaviour need not always yield optimal schedules and it has been observed that most often it is sub-optimal (Klemmt et al. 2009). To achieve the lowest makespan possible, it may be essential to let a machine remain idle for an interval of time, even though a job is available for processing on it. After some time, another job is allowed to seize the machine, ahead of the other jobs waiting for it in the queue. This behaviour is called as 'delay' scheduling.

Some additional modelling is required to modify the default non-delay behaviour of the simulation model and allow delay schedules. We introduce an infinite capacity delay module called as the controller delay and force every job to visit this buffer before proceeding to the next machine in its sequence. A value of zero for delay indicates that the job is to be sent immediately to the next machine.

5 OPTIMIZATION FORMULATION

Since the literature survey identifies Manne's MIP model (Manne 1959) as the best model for JSSP, it is chosen as the starting point of the optimization part of the proposed scheme. The simulation and optimization models incorporate different details of the system, with some overlap. The goal of the hybrid formulation is to create a reduced solution space, so that scheduling problem can be addressed in a more computationally efficient manner, with minimum compromise on the quality of solutions.

Simulation model covers the operational logic of the job shop, that is, the queues, resource sharing etc. Hence the optimization model does not include any constraints related to job handling, sequencing and queuing. The constraints that give a lower bound to the makespan are retained. However, they are modified to include the same variables used in the simulation model. The value added time in the system is called as the processing time. The non-value added time is the time that a job spends waiting for the resource to get free. All such non-value added times are called delays.

Table 1 lists the notations used in the formulations.

Table 1: Parameters of proposed model.

Notation	Definition
n	number of jobs
m	number of machines
p_{ij}	processing time of job *j* on machine *i* (minute)
S_j	Order in which job *j* visits machines (non-repeating set)
q_{ij}	priority of job *j* on machine *i* (ranks)
$\hat{d}_{ij}$	controller delay for job *j* before machine *i* (minute)
$\tilde{d}^r_{ij}$	observed wait time of job *j* before machine *i* from r^{th} sim. run (minute)
d^r_{ij}	suggested wait time of job *j* before machine *i* in r^{th} opt. run (minute)
$Cmax^r_{sim}$	makespan from r^{th} sim. run (minute)
$Cmax^r_{opt}$	makespan from r^{th} opt. run (minute)

5.1 Optimization Model

As mentioned earlier, the objective of the problem is to minimize the makespan or the maximal completion time of the job shop. Now, the actual system makespan is evaluated by the DES model, since it incorporates the operational logic. In the optimization model, a proxy objective, similar to makespan is used. This value is not the actual makespan of the job shop,since the resource sharing is accounted for in the optimization model. *Cmax* is the time when the last job finishes in the system. The objective is to minimize this value. Expression 1 gives the objective function for the LP.

There are three sets of constraints with two types in each set (one for jobs and one for machines). The first set gives the lower bound for makespan as shown in constraints 2 and 3. The second set, giving the lower bound for non-value added time in terms of $\tilde{d}$ is shown by constraints 4 and 5. The lower bound is some fraction $\frac{1}{k}$ of $\tilde{d}$, the delays observed in the previous simulation run. The output is sensitive to the choice of *k*.

The constant *k* is used in the optimization routine to constrain the non-value added time from optimization (which is the output) with respect to the non-value added time from simulation. This constant can be defined differently for the constraints over jobs and machines or the same value of *k* may be used for both. Also, *k* may be static (pre-defined) for all iterations or dynamic (changes with each iteration).

The formulation of the LP is presented next. The ISO model is tested on three benchmark instances from the OR library, representing small, medium and large sizes of job shops. The domains for the decision variables are decided based on the problem size. These domains and the values of parameter k giving the best performance for each model size are noted below.

Decision variable domains

`ft06`	$D = 30$, $k_m = 1.2$, $k_j = 2.5$
`ft08`	$D = 80$, $k_m = 2$, $k_j = 4$
`la26`	$D = 180$, $k_m = 5$, $k_j = 8$

$$\min_{d_{ij}^r} Cmax^r \tag{1}$$

$$\sum_{j=1}^{n} \{d_{ij}^r + \frac{p_{ij}}{10}\} \leq Cmax^r \quad i = 1,...,m \tag{2}$$

$$\sum_{i=1}^{m} \{d_{ij}^r + \frac{p_{ij}}{2}\} \leq Cmax^r \quad j = 1,...,n \tag{3}$$

$$k_m * \sum_{j=1}^{n} d_{ij}^r \geq \sum_{j=1}^{n} \tilde{d}_{ij} \quad i = 1,...,m \tag{4}$$

$$k_j * \sum_{i=1}^{m} d_{ij}^r \geq \sum_{i=1}^{m} \tilde{d}_{ij} \quad j = 1,...,n \tag{5}$$

$$d_{ij}^r \leq D \quad i = 1,...,m, j = 1,...,n \tag{6}$$

5.2 ISO Algorithm

The algorithm is run for a fixed number of iterations (R). First, the simulation model is run by using $q_{ij}^r = 0$ and $\hat{d}_{ij}^r = 0$. The delays from the simulation model $\hat{d}_{ij}^r$ and the makespan $Cmax^r$ are noted for each iteration r. These values are given to the optimization model to be used in the updating constraints. The optimization model returns the values for d_{ij}^{r+1} which are used as controller delays in simulation. The queue priorities are derived using these controller delay values. At the end of R iterations, the lowest makespan and corresponding delays are returned.

Algorithm

```
1: Set $q_{ij}^r = 0$ and $\hat{d}_{ij}^r = 0$.
2: **while** $r \leq R$ **do**
3:     Run simulation model. Note $\tilde{d}_{ij}^r$.
4:     Run optimization model with $\tilde{d}_{ij}^r$ as input. Note $d_{ij}^r$.
5:     Derive $q_{ij}^r$ using $d_{ij}^r$.
6:     $r = r + 1$
7: **end while**
8: return $d^R$
```

To demonstrate the algorithm, the values for all parameters for the first two iterations are recorded and presented for the `ft08` instance in Tables 2 to 5. As mentioned in the first three steps of the algorithm, the simulation model is first run (r = 0) with inputs $q_{ij}^0 = \hat{d}_{ij}^0 = 0$. The actual delays observed in the simulation model ($\tilde{d}_{ij}^0$) are noted in Table 2. These values are given as input to the optimization model as mentioned in step 4. The output of the optimization model (d_{ij}^0) is noted in Table 3. In step 5, the d_{ij}^0 values are assigned as queue priorities and controller delays. This completes one iteration. The queue priorities are set at 'smaller-the-higher', that is, a smaller value indicates a higher queue priority. The simulation model is run again (r = 1) using these new inputs ($q^1_{ij} = \hat{d}^1_{ij} = d^0_{ij}$) to obtain the output ($\tilde{d}^1_{ij}$) mentioned in Table 4. These values are used as input to the optimization model to obtain the next set of results mentioned in Table 5. This process continues iteratively for a specified number of runs at the end of which the best simulation makespan is returned as the final output, along with the values of corresponding d_{ij}.

Table 2: Simulation output $\tilde{d}^0_{ij}$, makespan: 1028.

43	0	0	0	66	0	112	41
146	0	127	46	143	181	0	140
31	86	0	92	115	31	58	0
75	37	0	0	36	97	0	0
69	0	0	0	0	14	0	60
0	21	46	0	79	0	65	6
28	0	30	1	0	0	0	0
85	0	62	19	42	0	41	41
0	90	0	101	107	105	29	91
31	0	60	11	47	0	0	53

Table 3: Optimization output d^0_{ij}, objective: 445.

0	80	1	20	30	0	0	0
2	80	80	80	80	70	0	0
0	29	80	18	80	0	0	0
65	0	0	0	58	0	0	0
0	0	0	0	0	72	0	0
0	0	0	0	0	32	77	0
0	0	0	0	0	0	15	15
80	0	0	0	0	0	65	0
80	0	0	0	0	22	80	80
21	0	0	0	0	0	0	80

Table 4: Simulation output $\tilde{d}^1_{ij}$, makespan: 1069.

0	74	0	0	38	0	71	0
15	0	90	183	61	76	0	7
5	2	0	0	35	31	0	0
12	59	95	0	52	0	0	0
25	7	0	0	71	0	4	0
22	0	0	0	13	0	6	0
21	61	18	63	0	0	0	0
0	95	0	23	1	8	2	0
141	0	0	27	9	14	35	0
50	11	14	0	0	0	0	40

Table 5: Optimization output d^1_{ij}, objective: 401.

0	78	55	18	0	0	0	0
80	0	0	56	80	0	0	0
0	0	0	0	0	25	12	0
0	0	0	0	61	48	0	0
0	0	0	0	0	0	34	20
0	0	0	0	0	0	0	21
0	0	0	0	0	80	2	0
0	0	0	0	0	0	65	0
0	0	0	0	0	0	80	33
0	0	0	0	0	0	0	58

6 RESULTS

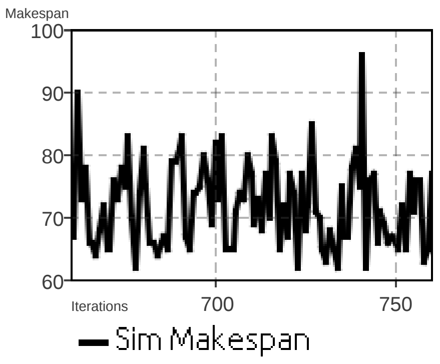
The experiments were performed on the 6×6 benchmark instance `ft06`, the 8×10 benchmark instance `ft08` and the 20×10 benchmark instance `la26`. The machine used had the following specification: Intel Dual Core, 800 MHz, with 3GB RAM. The software used in the implementation are Anylogic: 6.9, Eclipse: Indigo, CPLEX 12.4.

Using the values of k mentioned in Table 6, the results in the subsequent columns were obtained. Each experiment was run for 5000 iterations which took less than 100 sec to terminate.

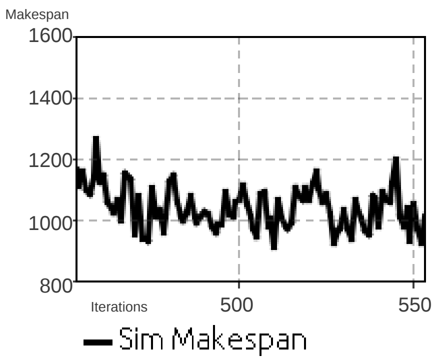
Table 6: Results for ISO.

Data instance	km	kj	Makespan	Time(sec)
`ft06`	1.2	2.5	58	60
`ft08`	2	4	869	60
`la26`	5	8	1347	82

The simulation model evaluates a schedule based on the values of decision variables suggested by the LP. Figure 3 shows the different makespans evaluated by the simulation model for the two benchmark instances `ft06` and `ft08`. The X-axis is the iteration number and the Y-axis the makespan. The figures show the different makespans evaluated in a duration of 100 iterations out of the total 5000 iterations performed. It is observed that although ISO may find a 'good' solution early on, it will still evaluate other makespans, which may be worse than a previously obtained solution, in order to try to move towards a better solution. The variability in the makespans over iterations, as seen in Figure 3, is due to the way ISO operates in order to escape local optima and explore the the entire solution space in a random manner.



(a) `ft06`.



(b) `ft08`.

Figure 3: Makespan as obtained from simulation in ISO method for 6×6 and 8×10.

The MIP methods are known to guarantee optimality for smaller problems. On comparing the results from ISO with the MIP method(Manne 1959), it is seen that the MIP achieves optimal solution in a few seconds for smaller problems. However, for the larger problem, the performance of the ISO is better than the performance of MIP running for 30 minutes.

Table 7: Comparison with MIP.

	MIP		ISO	
Data instance	Makespan (5min)	Makespan (30min)	Makespan	Time (sec)
`ft06`	55*	55*	58	60
`ft08`	824*	824*	869	60
`la26`	1372	1359	1347	82

*indicates optimal

The results indicate a potential for the proposed method, especially for larger problems.

7 CONCLUSIONS AND FUTURE RESEARCH

We proposed and presented an iterative solution approach to address the classical job shop scheduling problem with the objective of makespan minimization. The ISO approach is time efficient, has the flexibility of adding more information through extra constraints and there is no inherent randomness in the solution method itself, ensuring repeatability of experiments. Preliminary experiments have shown that the ISO is computationally faster than MIP in reaching a 'good' solution for larger problems.

This work is a part of ongoing research in hybrid solution approaches to the job shop scheduling problem. The method proposed in this paper will be benchmarked with the popular simulation-based optimization approach. Detailed analysis of the effect of parameter *k* will be carried out. Different ways of using the outputs from the optimization model in the simulation model will be explored. For example, the output from optimization can be mapped to the simulation model using intermediate processing to come up with values of q_{ij} and $\hat{d}_{ij}$. The algorithm will be tested on more benchmark instances from the OR library

A APPENDICES

Each instance in the benchmark library consists of a line of description followed by a line containing the number of jobs and the number of machines. Next, is one line for each job, listing the machine number and processing time for each step of the job. The machines are numbered starting with 0. The following is the ft10 instance.

Table 8: Fisher and Thompson 10x10 instance, alternate name (mt10)

10	10																		
0	29	1	78	2	9	3	36	4	49	5	11	6	62	7	56	8	44	9	21
0	43	2	90	4	75	9	11	3	69	1	28	6	46	5	46	7	72	8	30
1	91	0	85	3	39	2	74	8	90	5	10	7	12	6	89	9	45	4	33
1	81	2	95	0	71	4	99	6	9	8	52	7	85	3	98	9	22	5	43
2	14	0	6	1	22	5	61	3	26	4	69	8	21	7	49	9	72	6	53
2	84	1	2	5	52	3	95	8	48	9	72	0	47	6	65	4	6	7	25
1	46	0	37	3	61	2	13	6	32	5	21	9	32	8	89	7	30	4	55
2	31	0	86	1	46	5	74	4	32	6	88	8	19	9	48	7	36	3	79
0	76	1	69	3	76	5	51	2	85	9	11	6	40	7	89	4	26	8	74
1	85	0	13	2	61	6	7	8	64	9	76	5	47	3	52	4	90	7	45

Klemmt et.al.(Klemmt, Horn, Weigert, and Wolter 2009) have modified the above dataset to use an *8x10* job shop. This dataset is called `ft08` and is obtained by removing the last two rows of the above table. From Table 8, sorting out the data into processing time and sequence:

Table 9: Processing times for Fisher and Thompson 8x10 instance

29	78	9	36	49	11	62	56	44	21
43	28	90	69	75	46	46	72	30	11
85	91	74	39	33	10	89	12	90	45
71	81	95	98	99	43	9	85	52	22
6	22	14	26	69	61	53	49	21	72
47	2	84	95	6	52	65	25	48	72
37	46	13	61	55	21	32	30	89	32
86	46	31	79	32	74	88	36	19	48

The functions written in the model are:

- InitializeJob(): All entities are initialized by assigning their respective sequences and the priority for the first machine that is visited in their sequence.
- AssignControllerDelay(): When implementing delay schedules, this function is used to delay a job before it proceeds to the next machine in its sequence. For non-delay schedules, the function returns a value zero.
- routing(): This function is used to implement the sequences for each job. Each time a job leaves a machine, this function is called to route it to the next machine in its sequence.
- AssignInterDelay(): This function is used to assign the queue priorities of each job on each machine. This function is called each time after a new routing occurs.

Table 10: Sequences for Fisher and Thompson 8x10 instance

1	2	3	4	5	6	7	8	9	10
1	3	5	10	4	2	7	6	8	9
2	1	4	3	9	6	8	7	10	5
2	3	1	5	7	9	8	4	10	6
3	1	2	6	4	5	9	8	10	7
3	2	6	4	9	10	1	7	5	8
2	1	4	3	7	6	10	9	8	5
3	1	2	6	5	7	9	10	8	4

- ControllerDelayTimes(): This function calculates the time an entity has spent before entering the queue of the next machine in its sequence.
- DelayTimes(): This function calculates the time spent by the in the queue of a machine.
- BeforeProcessing(): This function is called just before a job seizes a machine resource. It notes the actual start time of servicing by calling function ActualStartTimes() and also calls DelayTimes().
- AfterProcessing(): This function is called after a job releases the machine resource. It notes the time at which the machine was released by calling the function EndTimes(). It also calls JobSequence() to read the next machine number and AssignInterDelay().
- AssignProcessTime(): This function is called once the job enters the delay of the machine. The processing time corresponding to the job and the machine is accessed from the ”processingtimes” table, which holds all the processing times.

REFERENCES

Bang, J.-Y., and Y.-D. Kim. 2010. “Hierarchical Production Planning for Semiconductor Wafer Fabrication Based on Linear Programming and Discrete-Event Simulation”. *IEEE Transactions on Automation Science and Engineering* 7 (2): 326–336.

Byrne, M. D., and M. A. Bakir. 1999. “Production planning using a hybrid simulation - analytical approach”. *International Journal of Production Economics* 59 (1-3): 305–311.

Glowacka, K. J., R. M. Henry, and J. H. May. 2009. “A hybrid data mining/simulation approach for modelling outpatient no-shows in clinic scheduling”. *Journal of the Operational Research Society* 60 (8): 1056–1068.

Jeong, S. J., S. J. Lim, and K. S. Kim. 2006. “Hybrid approach to production scheduling using genetic algorithm and simulation”. *International Journal of Advanced Manufacturing Technology* 28:129–136.

Klemmt, A., S. Horn, G. Weigert, and K.-J. Wolter. 2009. “Simulation-based optimization vs. mathematical programming: A hybrid approach for optimizing scheduling problems”. *Robotics and Computer-Integrated Manufacturing* 25:917–925.

Lee, Y. H., and S. H. Kim. 2000. “Optimal production-distribution planning in supply chain management using a hybrid simulation-analytic approach”. In *Proceedings of the 2000 Winter Simulation Conference*, edited by J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 1252–1259. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Mahdavi, I., B. Shirazi, and M. Solimanpur. 2010. “Development of a simulation-based decision support system for controlling stochastic flexible job shop manufacturing systems”. *Simulation Modelling Practice and Theory* 18 (6): 768 – 786.

Manne, A. S. 1959. “On the Job Shop Scheduling Problem”. Cowles Foundation Discussion Papers 73, Cowles Foundation for Research in Economics, Yale University.

Matta, A. 2008. “Simulation Optimization with Mathematical Programming Representation of Discrete Event Systems”. In *Proceedings of the 2008 Winter Simulation Conference*, edited by S. J. Mason,

R. R. Hill, L. Mönch, O. Rose, T. Jefferson, and J. W. Fowler, 1393–1400. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

McMullen, P. R., and G. V. Frazier. 1999. "Using Simulation and Data Envelopment Analysis to Compare Assembly Line Balancing Solutions". *Journal of Productivity Analysis* 11 (2): 149–168.

OR-library 2014. "OR library: job shop benchmark instances". Accessed Apr. 15, 2014. http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt.

Pan, J.-H., and J.-S. Chen. 2003. "Minimizing makespan in re-entrant permutation flow-shops". *Journal of the Operational Research Society* 54 (6): 642–653.

Schruben, L. W. 2000. "Mathematical Programming Models of Discrete Event System Dynamics". In *Proceedings of the 2000 Winter Simulation Conference*, edited by J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 381–385. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.: Society for Computer Simulation International.

van Laarhoven, P. J. M., E. H. L. Aarts, and J. K. Lenstra. 1992, January. "Job Shop Scheduling by Simulated Annealing". *Operations Research* 40 (1): 113–125.

AUTHOR BIOGRAPHIES

KETKI KULKARNI is a Ph.D. candidate and Research Scholar in the Industrial Engineering and Operations Research department at Indian Institute of Technology Bombay in Mumbai, Maharashtra India. She holds a M.S. in Industrial Engineering from the Georgia Institute of Technology, Atlanta, USA. Her email address is ketki.k@iitb.ac.in.

JAYENDRAN VENKATESWARAN is an Assistant Professor of Industrial Engineering and Operations Research at Indian Institute of Technology Bombay. He holds M.S. and Ph.D. degrees in Systems and Industrial Engineering from University of Arizona, Tucson. His research and teaching interests are primarily in the areas of modelling, simulation and analysis of complex systems (viz. supply chains). His email address is jayendran@iitb.ac.in.