Abstract: In this paper, a new model of capacitated lot sizing and scheduling in a permutation flow shop is developed. In this model demand can be totally backlogged. Setups can be carryover and are sequence-dependent. It is well-known from literatures that capacitated lot sizing problem in permutation flow shop systems is NP-hard. This means the model is solved in polynomial time and metaheuristics algorithms are capable of solving these problems within reasonable computing load. Metaheuristic algorithms find more applications in recent researches. On this concern this paper proposes two evolutionary algorithms, one of the most popular namely, Genetic Algorithm (GA) and one of the most powerful population base algorithms namely, Imperialist Competitive Algorithm (ICA). The proposed algorithms are calibrate by Taguchi method and be compared against a presented lower bound. Some numerical examples are solved by both the algorithms and the lower bound. The quality of solution obtained by the proposed algorithm showed superiority of ICA to GA.

Keywords: permutation flow shop; evolutionary algorithms; lot sizing and scheduling

1. INTRODUCTION

The multilevel lot sizing problem concerns how to determine the lot size for producing or procuring an item at each and sequencing is to determine job ordering on each level. The objective of lot sizing and sequencing generally is to minimize the sum of the total setup cost and inventory holding cost. Lot sizing and sequencing problem plays an important role in the efficient operation of modern manufacturing and assembly processes.

The flow shop lot sizing problem has been a very extensively researched area since the seminal paper of Johnson [1]. In the flowshop problem (FSP) a set of unrelated jobs are to be processed on a set of machines. These machines are disposed in a sequence and each job has to visit all of them in the same order. A special case of flow shop that assumes the same order of products in all machines is called permutation flow shop. In this paper we consider a permutation flow shop problem with setup carryover, setup sequence-dependent and backlogging.

In highly capacitated environments as well as in many real-life situations, the inclusion of back orders is crucial because otherwise, no feasible plan would exist and the respective result that no feasible solution can be found is of minor importance in practical settings. On the other hand, in many real-life manufacturing environment the capacity of the machines are limited, or for cost saving reasons, it might be useful to produce a product volume in a period other than its demand period to save setup time and costs. In traditional lot sizing models producing of a product in a period before its delivery to the customer is permitted. In this case, inventory cost occurs. In our case, it is also possible that the product cannot be delivered on time. It is then backlogging occurs and backlogging costs are incurred for every unit at period of the delay. While only few lot sizing approaches consider the possibility of back ordering, it is of great importance in practical settings: If capacity is limited, some products may have to be backlogged [2].

Quadt and Kuhn [3] investigated a capacitated lot sizing and scheduling problem with setup times, setup carryover, backorders, and parallel machines. They formulated a mixed integer formulation of the problem and a new solution procedure. The solution procedure was based on a novel "aggregate model" which uses integer instead of binary variables. Song and Chan [4] considered a single item lot sizing problem with backlogging on a single machine at a finite production rate. The objective function was to minimize the total cost of setup, stockholding and backlogging to satisfy a sequence of discrete demands. Other researchers have considered backlogging including Wolsey and Pochet [5]. Cheng et al. [6], and Karimi et al. [7].

Graham et al. [8] showed that the permutation flow shop scheduling problem is strongly NP-complete. Since then many researches have been attracted to develop heuristic and metaheuristic algorithms for these problems. Among researchers that employed metaheuristic we can cite to Mohammadi et al. [9]. They employed a GA as a solution approach. Their proposed algorithm was used for a simultaneous lot sizing and sequencing problem in permutation flow shops involving sequence-dependent setups and capacity constraints. Ruiz et al. [10] proposed new genetic algorithms for solving the permutation flow shop scheduling. The minimization of the total completion time or makespan was considered as the optimization criterion. They used new genetic operators, advanced techniques like hybridization with local search and an efficient population initialization as well as a new generational scheme. The
2. PROBLEM FORMULATION

The following notations are used in the model:

2.1. Notations and assumptions

2.1.1 Indices

- \( i, j, k \): Index of production type
- \( n \): Index of product type
- \( n' \): Designation for a specific setup number
- \( m \): Index of level of production
- \( t \): Index of period

2.1.2 Parameters

- \( T \): Planning horizon
- \( N \): Number of different products
- \( M \): Number of production levels/number of machines
- \( b_{gM} \): A large real number
- \( c_{m,t} \): Available capacity of machine \( m \) in period \( t \) (in time units)
- \( d_{j,t} \): External demand for product \( j \) at the end of period \( t \) (in units of quantity)
- \( h^{+}_{j,m} \): Storage costs unit rate for product \( j \) in level \( m \).
- \( h^{-}_{j,t} \): Shortage costs unit rate for product \( j \) at the end of period \( t \).
- \( b_{j,m} \): Capacity of machine \( m \) required to produce a unit of product (or shadow product) \( j \) (in time units per quantity units).
- \( P_{j,m,t} \): Production costs to produce one unit of product \( j \) on machine \( m \) at period \( t \) (in money unit per quantity unit).
- \( S_{i,j,m} \): Sequence-dependent setup time for the setup of the machine \( m \) from production of product \( i \) to production of product \( j \) (in time units); for \( i \neq j \), \( S_{i,j,m} \geq \text{band} = j, S_{i,j,m} = 0 \).
- \( W_{i,j,m} \): Sequence-dependent setup cost for the setup of the machine \( m \) from production of product \( i \) to production of product \( j \) (in money units); for \( i \neq j \), \( W_{i,j,m} \geq 0 \) and \( i = j, W_{i,j,m} = 0 \).
- \( J_{om} \): The starting setup configuration on machine \( m \).

2.1.3 Decision variables

- \( l^{+}_{j,m,t} \): Stock of product \( j \) at level \( m \) at the end of period \( t \).
- \( l^{-}_{j,t} \): Shortage of product \( j \) at the end of period \( t \).
- \( y^{n}_{i,j,t} \): Binary variable, which indicates whether the \( nth \) setup on machines at period \( t \) is from product \( i \) to product \( j \) (\( y^{n}_{i,j,t} = 1 \)) or not (\( y^{n}_{i,j,t} = 0 \)).
- \( x^{n}_{j,m} \): Quantity of product \( j \) produced after \( nth \) setup on machine \( m \) at period \( t \).
objective function is to find an optimal lot sizing and sequencing that minimize setup, inventory, production and backlogging costs.

\[
\min \sum_{n=1}^{N} \sum_{j=1}^{T} W_{i,j,m} Y_{i,j}^{n} + \sum_{n=1}^{N} \sum_{j=1}^{T} P_{j,m,t} X_{j,m}^{n} + \sum_{j=1}^{M} \sum_{m=1}^{T} h_{j,m}^{n} - I_{j,m}^{n} + \sum_{j=1}^{M} \sum_{t=1}^{T} h_{j,t}^{n} - L_{j,t}^{n} \tag{1}
\]

Subject to

\[
d_{i,j} = I_{j,m,t-1}^{n} + \sum_{n=1}^{N} x_{i,j,m}^{n} - l_{j,m,t}^{n} - l_{j,t-1}^{n} + l_{j,t}^{n}; \quad j = 1, \ldots, N, \quad t = 1, \ldots, T \tag{2}
\]

\[
l_{j,m,t}^{n} + \sum_{n=1}^{N} x_{j,m}^{n} + \sum_{n=1}^{N} b_{j,m} - q_{j,m,t}^{n} + \sum_{n=1}^{N} b_{j,m} - x_{j,m,t}^{n} \leq c_{m,t}; \quad m = 1, \ldots, M, \quad T \tag{3}
\]

\[
y_{i,j,m}^{n} \leq \left( \frac{c_{m,t}}{b_{j,m}} \right) \sum_{n=1}^{N} y_{i,j,t}^{n}; \quad n = 1, \ldots, N, \quad j = 1, \ldots, N, \quad m = 1, \ldots, M, \quad t = 1, \ldots, T \tag{4}
\]

\[
y_{j,t}^{n} = 0 \quad j \neq j_{m}, \quad i = 1, \ldots, N \tag{5}
\]

\[
\sum_{i=1}^{N} y_{j,m,i}^{n} = 1, \quad j = 1, \ldots, N \tag{6}
\]

\[
\sum_{i=1}^{N} y_{j,t,i}^{n} = \sum_{k=1}^{N} y_{j,t,k}^{n+1}; \quad i = 1, \ldots, N, \quad n = 1, \ldots, N - 1, \quad m = 1, \ldots, M, \quad t = 1, \ldots, T \tag{7}
\]

\[
y_{i,j,t}^{n} = 0 \text{ or } 1 \tag{8}
\]
In this model, the objective function is Equation (1). The backlogging or storage at the end of each period is considered by Equation (2). Constrains (3) ensures total of in-flows to each node is equal to of out-flows from that node. Equation (4) ensures within one period each typical product \( j \) one machine \( m \) is produced before its direct successor. The capacity constraints of machines are considered by Equation (5). Equation (6) respects setups in production process. Equation (7) indicates the relationship between shadow products and setups. Constraints (8) and (9) ensure that for each machine, the first setup at the beginning of the planning horizon is from a defined product. Equation (10) represents the relationship between successive setups. The type of variables is defined by Equations (11) and (12) and finally Equation (13) indicates that at the end of planning horizon there is no on-hand inventory.

2.3. Lower bound

In this section we present a lower bound that developed by Mohammadi et al. [27]. We first relax binary variables to continuous variables that fall in \([0, 1]\).

Then we add following equation to relaxed model:

\[
\sum_{i=1}^{N} y^i_{j,t} + \sum_{i=1}^{N} y^i_{j,t+1} = a_{j,t} \quad \text{(14)}
\]

In this equation \( a_{j,t} \) is binary variable.

Equation (14) was proved that is valid to model. We refer the proof of this equation to Mohammadi et al [27].

3. GENETIC ALGORITHM

GAs are probabilistic search optimization algorithms that were inspired by the process of natural evolution and the principles of survival of the fittest [28]. Genetic Algorithms (GAs) can be employed to find a near-optimal solution for \( NP \)-hard problems. GAs evolves a population of individuals according to the progress of algorithm to reach a good solution. Genetic operators (such as natural selection, mutation, and cross over) manipulate individuals in a population of solution over several iterations to improve their fitness. The algorithm generates a new candidate pool of solutions iteratively from the presently available solutions and replaces some or all of the existing members of the current solution tool with the newly created feasible solutions.

We first present a simple and effective heuristic to generate initial solution and then discuss the issue of encoding in our case an integer array representation. We then turn to the evolutionary stages of the algorithm and the specific genetic operators that have been designed to increase search efficiency.
Selection operator

The requirement parents for using of crossover have been obtained by one of the five selection method, Deterministic Sampling (A), Random Sampling (B), Roulette Wheel (C), Ranking (D) and Tournament (E).

Crossover operation and mutation operator

Several crossover operators have been proposed in reference [29]. Similar job two point crossover has been used in this research. In order to produce small perturbations on chromosomes to promote diversity of the population, a shift mutation operator has been used in this article. Crossover and mutation probability must be determined during parameters calibration.

Population replacement

Chromosomes for the next generation are selected from the enlarged population. The best pop_size chromosomes of the enlarged population have been selected for the next generation.

Termination criterion

The algorithm must terminate according to a criterion. This criterion is specified by reaching to maximum number of iteration $it_{max}$.

4. IMPERIALIST COMPETITIVE ALGORITHM

ICA is a novel population-based evolutionary algorithm proposed by Atashpaz-Gargari and Lucas [15]. The ICA initiates with an initial population, like most evolutionary algorithms. Each individual of the population is called a ‘country’ equivalent ‘chromosome’ in GA. Some of the most powerful countries are chosen to be the imperialist states and the other countries constitute the colonies of these imperialists. All the colonies of initial countries are partitioned among the mentioned imperialists based on their power. Equivalent of fitness value in the GA, the power of each country, is conversely proportional to its cost. An empire is constituted from the imperialist states with their colonies [30].

After all empires were formed, the competition between countries starts. First, the colonies in each of empires start moving toward their imperialist. During this movement, if the colony gets better cost function than its imperialist does, they will exchange their positions and the algorithm will continue with the new imperialist. The power of each empire is calculated by imperialist cost function and colonies. The empire which is weaker than the others loses its colonies. Each imperialist attempts to gain the colonies of other empires. The most powerful empires have a more chance to gain the colonies from the weakest empires. The more powerful an empire is, the more likely it will possess the weakest colony of the weakest empire (Imperialistic competition)

During the competition weak imperialists will lose their weakest colony gradually. When an empire loses all of its colonies, it will be eliminated from the population. In fact the empire collapses. The final level of imperialist rivalry is when there is only one empire in the world. The main steps of ICA are described as follows:

Step 1 Generating of Initial countries

Each individual of the population is called a ‘country’ equivalent ‘chromosome’ in GA. Each country denotes a socio-political characteristic in that country such as culture, language, business, economic policy and etc. The socio-political characteristic in countries is the same different type of variables. There are two different types of variables, continuous variables ($x, q, I^c, I^l$) and binary variables ($y$). Each country consists of five variables, $x, q, I^c, I^l$ and $y$ where all of these variables must be optimized.

Initial values of continuous variables are generated randomly by uniform distribution function. To generate initial value for binary variable, we use a simple and effective heuristic which has been presented by Mohammadi et al. [27].

Step 2 Generating of Initial imperialists

A set of the most powerful countries form imperialists and the rest weaker countries are colonies of imperialists. The power of each country is calculated based on the objective function.

Step 3 Assimilation of colonies

Assimilation has been modeled by moving all the colonies toward the imperialist. Each country (colony) has different socio-political characteristics (variables), so every socio-political characteristic (variables) could move toward the related socio-political of imperialist in different ways. Continuous variables of colonies move toward related continuous variables of its imperialist and binary variables move toward binary variables of its imperialist.

The assimilation of continuous variables is modeled by moving the colony toward the imperialist by runits $x \sim U(0, \beta \times d)$. Where $\beta > 1$ and $d \sim U(\gamma, \rho)$. $d$ is distance between colony and the its imperialist.

The movement of binary variables is accomplished by crossover operation, like crossover operator in genetic algorithms. Crossover allows exchanging information between different solutions (chromosomes) so it is useful to assimilate binary variables.

Step 4 Revolution

The revolution increases the exploration of the algorithm and prevents the early convergence of countries to local minimums. A very high value of revolution decreases the exploitation power of algorithm and can reduce its convergence rate [31]. In each iteration, some of the colonies are chosen and their positions are exchanged. This mechanism is similar to mutation process in genetic algorithm for
creating diversification in solutions. Mutation increases the variety in the population, so this operator is used for creating a revolution in binary variables.

**Step 5 Exchange the colony with imperialist**

During assimilation and revolution, a colony may get to a situation with lower cost than the imperialist. In this case, the imperialist and the colony change their positions.

**Step 6 Imperialistic competition**

To start the competition, after selecting the weakest colony, the possession probability of each empire must be found. The normalized total cost of an empire is simply obtained by

$$NTC_n = TC_n - \max\{TC_i\}$$

Where, $NTC_n$ and $TC_n$ are the total cost and the normalized total cost of $n$th empire, respectively.

The total power of an empire is mainly contributed by the power of imperialist country. It is clear that the power of an empire includes the imperialist power and their colonies.

$$TC_n = cost(\text{imperialistar})C + \rho + \text{mean}[\text{cost}(\text{colonies of empire}_n)]$$

Where $\rho$ is a positive small number. The possession probability of each empire is given by

$$p_n = \frac{NTC_n}{\sum_{i=1}^{N} NTC_i}$$

Roulette wheel method was used for assigning the mentioned colony to empires.

**Step 7 Elimination of powerless empires.**

During the competition weak imperialists will lose their weakest colony gradually. When an empire loses all of its colonies, it collapses. At the end just one imperialist will remain. This is the optimum point.

**Step 8 Stop criterion**

In such an ideal new world, all the colonies will have the same positions and same costs and they will be controlled by an imperialist with the same position and cost as themselves. In such a world, there is no difference not only among colonies, but also between colonies and imperialist [32] in this situation; the algorithm has reached the global solution.

Stopping criterion in proposed algorithm is to get the maximum decades (maximum iteration).

5. PARAMETER CALIBRATION AND COMPUTATIONAL TESTING

www.ijsea.com
Table 1. Factor levels

<table>
<thead>
<tr>
<th>Parameter Level</th>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
<th>Level 4</th>
<th>Level 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection Type</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>Crossover Probability</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
<td>0.4</td>
<td>0.5</td>
</tr>
<tr>
<td>GA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mutation Probability</td>
<td>0.05</td>
<td>.10</td>
<td>.15</td>
<td>.20</td>
<td>.25</td>
</tr>
<tr>
<td>Number of Population</td>
<td>500/N</td>
<td>600/N</td>
<td>750/N</td>
<td>900/N</td>
<td>1000/N</td>
</tr>
<tr>
<td>Maximum Iteration</td>
<td>500/N</td>
<td>600/N</td>
<td>750/N</td>
<td>900/N</td>
<td>1000/N</td>
</tr>
<tr>
<td>ICA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Population</td>
<td>500/N</td>
<td>600/N</td>
<td>750/N</td>
<td>900/N</td>
<td>1000/N</td>
</tr>
<tr>
<td>Number of Imperialist</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>Revolution Probability</td>
<td>500/N</td>
<td>600/N</td>
<td>750/N</td>
<td>900/N</td>
<td>1000/N</td>
</tr>
<tr>
<td>$\rho$</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
<td>0.4</td>
<td>0.5</td>
</tr>
</tbody>
</table>

In this case, Taguchi method needs 25 experiments for each algorithm. As mentioned above, for each 25 experiment 15 independent runs are carried. The termination criterion of maximum elapsed CPU time is $t=7200S$ [9]. The required parameters for these problems are extracted from the following uniform distributions: $e \sim U(5,10)$, $d \sim U(0.5,1)$, $h^+ \sim U(0.05,0.1)$, $h^- \sim U(1.5), b \sim U(0.02,0.04), p \approx U(0.02,0.04), s \approx U(100,1100)$.

In order to conduct the experiments, we implemented GA and ICA examples in MATLAB run on a PC with a 2.27 GHz Intel Core i5 processor and 3 GB RAM memory and analyzed the result by Minitab 16 software. Figure 2 and 3 show the average S/N ratio obtained at each level for ICA and GA.

According to Figure 2 and Figure 3, the optimal levels of factors have been indicated in Table 2.

Table 2. The optimal levels of factors

Figure 2. Main effect plot for S/N ratios for GA
Figure 3. Main effect plot for S/N ratios for ICA
5.2. Comparison of the algorithms

In this section, in order to evaluate and compare the performance of two proposed, we consider different problem sizes.

For each problem set, 15 independent instances are randomly generated (225 problems) and the required parameters for these problems are extracted from the following uniform distributions:

\[ c \approx U(5, 10), d \approx U(0.5, 1), h^+ \approx U(0.05, 0.1), h^- \approx U(1, 5), b \approx U(0.02, 0.04), p \approx U(0.02, 0.04), s \approx U(100, 1100) \]

For each of the 15 instances, 15 independent runs are carried out for each algorithm within a reasonable CPU time, 7200 s. We obtain the mean of 15 instances as the response variable of each instance. This response is used to compare two algorithms. Problems have been solved in MATLAB run on a PC with a 2.27 GHz Intel Core i5 processor and 3 GB RAM memory. The computed results are reported in Table 3.

<table>
<thead>
<tr>
<th>Problem set</th>
<th>Dimension of problems ((N \times M \times T))</th>
<th>GA</th>
<th>ICA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 \times 2 \times 2</td>
<td>16.72%</td>
<td>9.66%</td>
</tr>
<tr>
<td>2</td>
<td>3 \times 3 \times 3</td>
<td>16.76%</td>
<td>9.42%</td>
</tr>
<tr>
<td>3</td>
<td>4 \times 4 \times 4</td>
<td>9.38%</td>
<td>6.46%</td>
</tr>
<tr>
<td>4</td>
<td>5 \times 5 \times 5</td>
<td>15.01%</td>
<td>11.69%</td>
</tr>
<tr>
<td>5</td>
<td>6 \times 6 \times 6</td>
<td>15.02%</td>
<td>8.55%</td>
</tr>
<tr>
<td>6</td>
<td>7 \times 7 \times 7</td>
<td>15.84%</td>
<td>7.48%</td>
</tr>
</tbody>
</table>
We are now employing a 95% confidence level and we are using Tukey HSD confidence intervals to compare algorithms with Minitab 16 Software. The result has been showed in Figure 4. From the Figure 4 it is clear that the proposed ICA algorithm is statistically better than the proposed GA algorithm.

Figure 4: Tukey HSD confidence intervals

6. CONCLUSION

This paper studies the permutation flow shop lo sizing and scheduling problem and developed a new model for the problem under sequence-dependent and carryover setups with considering backlogging.

To solve the problem, two metaheuristics was proposed namely, GA and ICA. Since the parameters of any algorithm has significant effect on algorithms performance, we use a fractional factorial experiment namely, Taguchi method. In order to evaluate the effectiveness and robustness of the proposed GA and ICA, we carried out a comparison between the algorithms. In this context, we presented a lower bound and compared the algorithms against it. The distance between the algorithms and the lower bound was calculated. Based on the results, Tukey HSD confidence intervals was employed to determine which algorithm is statistically superior to the other one. The results showed the ICA outperforms the GA.

As a direction for future research, it would be interesting to develop other metaheuristic, like Particle Swarm Optimization, Harmony Search and Honey Bee Algorithm. As an additional contribution, developing of the single objective into multi objective models can be considered.

REFERENCES

schedules with setup times included. 1954;1(61-68).


117-123.


[37] Onwubolu GC, Davendra D. Scheduling flow shops using differential evolution algorithm. 2006;171(2).