Efficient meta heuristic algorithms to minimize mean flow time in no-wait two stage flow shops with parallel and identical machines

Rasoul Shafaei, Nazanin Moradinasab*, Meysam Rabiee

Department of industrial engineering, K. N. Toosi University of Technology, Tehran 15875/4416, Iran

(Received 14 March 2011, Revised 7 April 2011, Accepted 29 June 2011)

Abstract. The no-wait two stage flexible manufacturing problem is an important problem in flow shops. This paper investigates the no-wait two stage flexible flow shop with a minimizing mean flow time performance measure. Six meta-heuristic algorithms are developed to solve the problem. In addition some numerical experiments are established to compare the efficiency of the proposed algorithms to each other. The results of the simulation study are illustrated to testify to the performance of the proposed algorithms. This is followed by proposing the most efficient algorithm and giving concluding remarks and potential areas for further research.

Keywords: no-wait, flexible flow shop, ICA, ACO, PSO

1 Introduction

Scheduling tasks are applied in many fields of industrial production as they intend to optimally utilize the resources while meeting customer requirements. Scheduling is considered the allocation of available resources to a set of operations or tasks over a planning horizon, the objective being to best satisfy one or more performance criteria, e.g. minimum make span, idle time or mean completion time (Naderi et al., 2011 [21]; Naderi et al., 2011 [20]).

Among the scheduling problems studied in the literature, the flow shop scheduling problem has been paid significant attention. One of the main reasons is that most of the batch type manufacturing systems fall into the flow shop category. The hybrid flow shop system combines the properties of the flow-shop and the parallel processor (machine). In the flow shop, there is only one processor at each stage. In the hybrid flow shop, there are one or more identical, uniform or unrelated parallel processors at each stage. The availability of more than one processor at some stages can offer additional flexibility for production scheduling and reduce production lead time. The hybrid flow shop scheduling problem (HFSP) was first proposed by Arthanari and Ramamurthy (1971) [3] and then attracted the attention of many researchers and practitioners (Allaoui and Artiba, 2004 [2]; Behnamian et al., 2010 [5]; Behnamian and Zandieh, 2011 [6]; Gholami et al., 2008 [9]; Karimi et al., 2010 [17]; Karimi et al., 2010 [16]; Zandieh and Gholami, 2009 [28]). This problem with two-stages and with each one having two identical machines (which is a simple and small problem) is considered NP-hard in the strong sense (Gupta, 1988 [12]).

In a flow shop, the scheduling problem can be classified into two categories namely with and without an operation interval waiting time. In a flow shop system with waiting times, the jobs are processed from one machine to the next one allowing waiting time in between, whereas, in a no-wait flow shop system, the jobs are processed from one machine to the next machine without waiting time. Therefore, in the classical flow shop sequencing problem with waiting time, jobs may be queued in front of each machine. In such a case, an unlimited buffer is considered at the front of each machine (Candar, 1999 [7]). In contrast, in a no-wait flow shop, jobs are processed from one machine to the next without waiting time. There are two main reasons for having a no-wait scheduling environment: either initiated from the nature of production or the lack of intermediate buffers. In some industries, due to the temperature or other attributes of the materials it is required that each operation follow the previous one immediately. This means, when necessary, the start of a job on a given machine is delayed in order that the operation’s completion coincides with the start of the next operation on the subsequent machine. Similarly, a no-wait flow shop aims at minimizing the in process buffer to obtain Just In Time production. Applications of a no-wait flow shop can be found in many industries such as plastic production processes that require a series of processes to immediately follow one after another in order to prevent material degradation during production. Similar situations also arise in the chemical and pharmaceutical industries (Aldowaisan and Allahverdi, 2004 [1]; Grabowski and Pempera, 2000 [11]; Hall and Sriskandarajah, 1996 [13]; Raaymakers and Hoogeveen, 2000 [23]). The no-wait problem has been extensively studied in the scheduling literature. Hall and Sriskandarajah (1996) [13] reviewed the literature on this subject. Reddi and Ramamurthy (1972) [24] proposed the Travelling Salesman Problem (TSP) technique to solve the flow shop scheduling problem. Gilmore and Gomory (1964) [10] also studied a two stage, single processor no-wait flow shop problem using the Travelling Salesman Problem (TSP) techniques. The results of the investigation revealed that a TSP based branch and bound algorithm

* Correspondence to: Tel.: +98-21 88678580, fax: +98-2188674858. E-mail address: N_moradinasab@sina.kntu.ac.ir.
obtained optimal solutions. In addition in a two stage no-wait flow shop with makespan performance, the proposed algorithm only required $O(n_2)$ steps to find an optimal solution.

![Fig. 1 Schematic of the problem](image)

With respect to the performance measure, the literature review shows that most researchers investigated the flexible flow shop scheduling in terms of the makespan performance measure. Among the other papers which applied a non-Makespan performance measure are a couple of papers presented by Jolai et al. (2009) [15] and Huang et al. (2009) [14] which are worthy to be reviewed. Jolai et al. developed a mixed integer linear programming model with the objective of maximizing the total profit gained from scheduled jobs. Furthermore, since their problem was NP-hard, an efficient genetic algorithm was presented as the solution procedure. Computational results showed that the presented approach outperforms the other algorithms in terms of both quality of solutions and required run times. Huang et al. (2009) [14] considered a no-wait two stage flexible flow shop with setup times and with a minimum total completion time performance measure. They proposed an integer programming model and Ant Colony Optimization heuristic approach. The ACO results revealed that the efficiency of the proposed algorithm is superior to those solved using integer programming.

In this paper, we investigate the no-wait two stage flexible flow shop scheduling problem with minimized mean flow time. The aim of this paper is to investigate the performance of the proposed meta-heuristic algorithms to solve a no-wait two stage flexible flow shop scheduling problem with minimized mean flow time. The remainder of this paper is organized as follows. In Section 2, the problem studied in this research is described in detail. In Section 3, the structure of the proposed algorithms is explained. Then numerical tests are established to solve the problems in section 4. This is followed by a demonstration of the simulation results. Finally Section 5 presents a summary of the research with concluding remarks and recommendations for further research.

2 Problem description

The problem studied in this paper is a no-wait two stage flexible flow shop scheduling problem. The performance of the proposed heuristic algorithm is studied in terms of the Minimization of mean flow time. The structure of the problems studied is as follows. A set of $n$ jobs $J = j_1, j_2, \ldots, j_n$ are to be processed in a flexible shop. Each job consists of two operations to be processed in two subsequent stages namely $S_1$ and $S_2$. No-waiting time is allowed between the two subsequent operations. Stages $S_1$ and $S_2$ have $m_1$ and $m_2$ identical machines, respectively. The processing times of job $j$ are $p_{ij}$ and $p_{ij}$ respectively. The problem is shown by $F_2(m_1, m_2)$ no-wait $[\tilde{F}]$. If the number of parallel machines in each stage are not input variables, the problem is shown by $F_2(P)$ no-wait $[\tilde{F}]$. In this study, the operation set up times are supposed to be independent from the job sequences and hence are added to the operation time. This problem is schematically showed in Fig. 1.

As illustrated above, each job has $m_1 \times m_2$ possible schedules and hence $n$ jobs have $n!$ possible schedules. Therefore in total there are $n! \times m_1^2 \times m_2^2$ possible solutions for this problem. The two stage no-wait flexible flow shop problems are NP-hard in the strong sense (Srisctandarajah and Ladet, 1986 [27]). In this paper, we propose six meta-heuristic algorithms for the above problem. The framework of these algorithms is described in the next section.

3 Proposed algorithms

In this section, we present six meta-heuristic algorithms. Before defining the algorithms, a fitness evaluation is applied to satisfy the no-wait constraint. This constraint is maintained by to postponement of the start time of those jobs where assigned machines are not available. The pseudo code of this step is shown in Fig. 2.

3.1 Imperialist competitive algorithm (ICA)

Atashpaz-Gargari and Lucas (2007) [4] illustrated the Imperialist Competitive Algorithm (ICA). The method is quite similar to other evolutionary algorithms using an initial population and any individual of the population is named a country. Countries are divided into two groups: imperialists and colonies. Some of the best countries (countries with the least cost) are chosen to be the imperialist countries and the rest are (i.e. colonies) divided among the mentioned imperialists based on imperialist power. The power of each country is calculated based on the objective function. A set of one imperialist and their colonies forms one empire. The total power of an empire is set equal to the power of the imperialist country plus a percentage of the mean power of its colonies. After forming the initial empires, the competition starts in a way that the colonies in each of empires start moving toward their imperialist country, and the imperialists attempt to gain more colonies. Therefore, during the competition, the weak imperialist collapses. At the end just one imperialist will remain.

In this paper we use two types ICA namely the Hybrid Imperialist Competitive Algorithm (HICA) and the Stochastic Imperialist Competitive Algorithm (SICA). After defining the job sequence, it is applied to both algorithms. The difference between these two algorithms is related to the assignment of jobs to machines at station one. The structure of these algorithms is described below.

3.1.1 Hybrid imperialist competitive algorithm (HICA)

In this algorithm the jobs are selected for the assignment based on the sequence already defined. The selected job is then assigned to a machine at the earliest available time. The code work of the HICA is as follows:

1. Generating initial empires

Each solution in a HICA is in the form of an array. Each array consists of a number of variables to be optimized. In GA terminology, this array is called a “chromosome,” but here, we use the term “country” for this array. In an $N$ di-
The initial number of colonies of an empire is calculated as follows:

\[
NC_n = \text{round}\{P_n, N_{col}\},
\]

where, \(P_n\) is the initial number of colonies of the \(n^{th}\) empire \(NC_n\) and assign them to each imperialist. The empires with the stronger power have more colonies than the empires with weaker power.

(2) Moving the colonies of an empire toward the imperialist (assimilating)

Colonies are improved by the imperialist countries by moving all colonies toward the imperialist. Therefore, at the end, some parts of a colony’s structure are similar to that of the empire’s structure. In a HICA the assimilating operator is performed as follows:

(1) Having generated the job sequences in the imperialist’s array, as shown in Fig. 4, a number, representing the selected job number, varying in range between 1 and total job number is generated randomly. Let’s assume that the selected job number is 5.

![Fig. 2 Pseudo code of no-wait](image)

<table>
<thead>
<tr>
<th>Imperialist</th>
<th>6</th>
<th>5</th>
<th>7</th>
<th>2</th>
<th>1</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colony</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

**Fig. 3** The structure of a solution for a problem with seven-jobs using HICA

The cost of a country is calculated using a cost function \(f\) at the variables \((P_1, P_2, P_3, \cdots, P_N)\) as follow:

\[
f = f(\text{country}) = f(P_1, P_2, P_3, \cdots, P_N). \quad (2)
\]

The algorithm starts with the generation of the initial countries with population size (PopSize) and the most powerful countries (countries with minimum costs) are selected as the imperialists. The remaining countries are colonies each of which belongs to an empire. The colonies are distributed among the imperialists based on imperialist power. The normalized cost of an imperialist is defined as follows:

\[
C_n = \max_i c_i - c_n, \quad (3)
\]

where, \(c_n\) is the cost of the \(n^{th}\) imperialist and \(c_n\) is its normalized cost. As a result the less normalized cost value originated by the imperialist has more cost value. The power of each imperialist is calculated and normalized as below and the colonies are distributed among the imperialists according to the power of each imperialist country.

\[
P_n = \left| \frac{C_n}{\sum_{i=1}^{N_{imp}} C_i} \right|. \quad (4)
\]

The initial number of colonies of an empire is calculated as follows:

**Fig. 4** Imperialist’s and colony’s arrays

(2) From the colony’s array, the selected job, i.e. job number 5, in the colony array as in Fig. 4 is moved to the same position as in the imperialist array while the subsequent positions are shifted accordingly. The resulting job positions are shown in Fig. 5.

| 7 | 5 | 2 | 4 | 3 | 1 | 6 |

**Fig. 5** Shifted colony

(3) Here the subsequent job from the imperialist selected job (i.e. job number 7) is chosen. Similar to step two, this job number is found in the shifted colony array (Fig. 5). Then this job number is swapped with the job number of the subsequent position of the selected job number from step one (i.e. swapping location number 7 with 2 in Fig. 5). The resulting shifted colony array is shown in Fig. 6.

| 2 | 5 | 7 | 4 | 3 | 1 | 6 |

**Fig. 6** Swap numbers in the shifted colony

| 2 | 5 | 7 | 4 | 1 | 3 | 6 |

**Fig. 7** Assimilated colony
Stochastic imperialist competitive algorithm

Ant colony algorithm

shows the pseudo code for this algo-

Elm. 11

follows:

colonies of an empire affects the total power of that em-

power of the imperialist country, but the power of the

perialist in its new position. The empire, after exchanging

TC

are shown in Fig. 7. This step is re-

peated based on a percentage of the total job number. The

percentage is a given parameter known as % Assimilate.

(3) Exchanging positions of the imperialist and a colony

A colony might reach a position with a lower cost than

the imperialist when the colony is moved toward the im-

perialist. In these situations, the position of the imperialist

and the colony is exchanged as shown in Fig. 8. After, the

algorithm continues and the colonies move toward the im-

perialist in its new position. The empire, after exchanging

the position of the imperialist and the colony, is depicted

in Fig. 8.

(4) Total power of an empire

The total power of an empire is mainly affected by the

power of the imperialist country, but the power of the

colonies of an empire affects the total power of that em-

pire. The resulting function of the total cost is defined as

follows:

\[ TC_n = \text{cost(}\text{imperialist}_n\text{)} + \xi \text{mean}\{\text{cost(}\text{colonies of empire}_n\text{)}\} \]  

where \( TC_n \) is the total cost of the \( n^{\text{th}} \) empire and zeta (\( \xi \)) is a positive number which is considered to be less than 1. The total power of the empire is determined only by the imperialist when the value of \( \xi \) is small. By increasing this value, the role of the colonies in determining the total power of an empire is increased.

(5) Imperialistic competition

All empires attempt to take possession and control of

the colonies of other empires. In imperialistic competition

the power of weaker empires is gradually reduced and the

power of the more powerful ones will rise. In other words,

selecting some (usually one) of the weakest colonies from

the weakest empire and generating competition among

the empires to possess these colonies is considered to be imperi-

alistic competition. In this competition, the most powerful

empires won’t definitely gain possession of these colonies,

but are more likely to than the weaker empires. The target

colony is selected from the weakest colony of the weakest

imperialist and then is available for the final lucky imperi-

alist. For this purpose the imperialist normalized total cost

is calculated using the following equation.

\[ NTC_n = \max\{TC_n\} - TC_n \]  

where, \( NTC_n \) is the normalized total cost of the \( n^{\text{th}} \) empire and is the total cost of the \( n^{\text{th}} \) empire. Having normalized the total cost, the possession probability of each empire is calculated as below:

\[ P_{pn} = \frac{NTC_n}{\sum_{i=1}^{N_{mp}} NTC_i} \]  

The lucky empire is selected using a Roulette wheel method and then a target colony is assigned to this empire.

(6) Revolution

In each iteration, some of the colonies are selected. Then

two positions of a colony’s array are chosen and these posi-

tions are exchanged. The replacement ratio is identified as

the revolution referred to as the P-Revolution.

(7) Eliminating the powerless empires

Powerless empires collapse and their colonies are dis-

dtributed among the other empires in imperialistic competi-

tion. In the research different collapse mechanisms are used to

define a powerless empire (Shokrollahpour et al., 2011

[26]). In this paper, when an empire loses all of its colonies,

it is considered a collapsed empire.

(8) Termination criteria

In this paper the termination criteria for imperialistic

competition occurs when there is only one empire for all of

the countries. Fig. 8 shows the pseudo code for this algo-

rithm.

3.1.2 Stochastic imperialist competitive algorithm
(SICA)

The structure of this algorithm is the same as the HICA
except in the generation of the initial empires and the mov-

ing of the colonies of an empire toward the imperialist (as-

similating), i.e. steps 1-2 and 4 are described as follows:

(1) Generating initial empires

The difference between HICA and SICA in generating

initial empires is in the structure of the established coun-

tries. The structure of a country in SICA is defined as fol-

lows:

In a SICA each country is a \( 1 \times 3N \) array of integer vari-

ables where \( N \) represents the number of jobs. In SICA a

country is formed from three parts. Each part is a \( 1 \times N \)

array. In this algorithm part one is similar to a country in

HICA that represents a sequence of jobs. In contrast with

HICA, in this algorithm the jobs are randomly assigned

to the machines in each stage. Part 2 represents the job as-

signment, obtained randomly, to the machines in stage one.

Part 3 shows the job assignment that is also obtained ran-

domly to the machines in stage two. The structure of one

solution for a problem with seven-jobs and two machines in

stage one and three machines in stage two is shown in Fig.

10.

As an example this figure shows that in stage one job

number 5 is in the second sequence on machine 1 and ma-

chine 2 in the stage two.

(2) Moving the colonies of an empire toward the imperialist

(assimilating)

In SICA, the assimilating operator in part 1 is similar to

HICA except for step 4. The difference between HICA and

SICA in assimilating is in part 2 and 3 of the SICA

assimilating operator which is defined as follows:

● A job number is randomly selected (e.g. Job number 7).

The machine to which it is assigned in the imperialist array

is chosen (i.e. machine number 1). Then in the colony array,

i.e. Fig. 11, the selected job number is assigned to the same

machine as in the imperialist array.

3.2 Ant colony algorithm

The logic behind the ant colony optimization (ACO) is to

simulate the riddling behavior of ant colonies. When a

group of ants set out from their nest to search for a food

source, they use a special kind of chemical, the pheromone,

to communicate with each other. Once the ants find a path

to the food source, they deposit pheromone on the path.
Hybrid ant colony optimization (HACO)

Colony (part 2)

Imperialist Colony


By feeling the pheromone on the ground, ants can follow the path to the food source discovered by other ants. As this process continues, most of the ants tend to choose the shortest path to the food as a large amount of pheromones have accumulated on this path. This collective pheromone-depositing and pheromone-following behavior of ants become the inspiring source of ACO [20].

In this paper, two algorithms based on ACO namely the Hybrid Ant Colony Optimization (HACO) and the Stochastic Ant Colony Optimization (SACO) are developed. The difference between these algorithms is related to the type of job assignment to the machines. In SACO, in both stages, the jobs are assigned to the machines randomly whereas in HACO the jobs are assigned to the machine at the earliest available time. The framework of HACO and SACO are described as follows:

3.2.1 Hybrid ant colony optimization (HACO)

In this algorithm the jobs are assigned to the machine at the earliest available time. The framework of HACO is as follows:

1. Initialization
   1.1- Set Parameters (PopSize, Number of imperialist, \( \xi \), P-Revolution, % Assimilate )
   1.2- Generating initial Countries (Randomly)
   2. Evaluate fitness of each country
   3. Form initial empires
   3.1- Choice power countries as imperialists
   3.2- Assign other countries (colonies) to imperialists based on their power
   4. Move the colonies of an empire toward the imperialist (assimilation)
   5. Revolution among colonies and imperialist
   6. If the cost of colony is lower than own imperialist
   6.1- Exchanging positions of the imperialist and a colony
   7. Calculate Total power of the empires.
   8. Imperialistic competition
   8.1- Select the weakest colony of the weakest empire and assign this to one of the strange empires
   9. Eliminate the powerless empires (the imperialist with no colony)
   10. Stop if stopping criteria is met, otherwise go to step 4.

Fig. 9 Pseudo code of HICA

Fig. 10 The structure of a solution for the problem with seven-jobs in SICA

Fig. 11 Assimilate colony in part 2 and 3

3.1.1.6. Revolution

In each iteration, some of the colonies are selected. Then two positions of colony’s array are chosen and these positions are exchanged. The replacement ratio is identified as the revolution referred as revolution among colonies and imperialist.

3.1.1.8. Termination criteria

In this paper the termination criteria for imperialistic competition is occurred when there is only one empire for all of the countries. Figure 9 shows pseudo code of this algorithm.

By feeling the pheromone on the ground, ants can follow the path to the food source discovered by other ants. As this process continues, most of the ants tend to choose the shortest path to the food as a large amount of pheromones have accumulated on this path. This collective pheromone-depositing and pheromone-following behavior of ants become the inspiring source of ACO [Chen et al., 2007 [8]].

In this paper, two algorithms based on ACO namely the Hybrid Ant Colony Optimization (HACO) and the Stochastic Ant Colony Optimization (SACO) are developed. The difference between these algorithms is related to the type of job assignment to the machines. In SACO, in both stages, the jobs are assigned to the machines randomly whereas in HACO the jobs are assigned to the machine at the earliest available time. The framework of HACO and SACO are described as follows:

3.2.1 Hybrid ant colony optimization (HACO)

In this algorithm the jobs are assigned to the machine at the earliest available time. The framework of HACO is as follow:

(1) Initialization of the algorithm

The parameters of HACO are defined as follows: \( N \) is the number of ants, \( \tau_0 \) is the initial pheromone trial, \( R_0 \) is a comparative proportional ratio of controlled exploitation and exploration, \( \rho \) the proportion of evaporated local pheromone, \( \alpha \) is the parameter that determines the proportion of global evaporation, \( \beta \) shows the importance of heuristic information, \( \tau(i, j) \) is the pheromone trial on the path between node \( i \) and node \( j \). Hence we denote the pheromone value associated with choosing job \( j \) after job \( i \) by \( \tau(i, j) \), \( \eta(i, j) \) that is defined as the inverse of the largest completion time among all feasible nodes.

(2) Initialization of ants

A group of \( N \) ants are initialized then all ants are set to the initial state at the beginning of each iteration. In a HACO, each ant represents a sequence of jobs to be assigned to a machine at the earliest available time in both stages. The structure of a solution for a problem with seven jobs, which is similar to HICA, is shown in Fig. 3.

(3) Solution construction

Each ant selects the first job randomly. It is followed by choosing the next job according to the rules defined in Eq. (9) and Eq. (10). It is continued until all of the jobs are selected. In order to select one of the rules defined in Eq. (9) and Eq. (10), a random number, say \( q \), is selected. If \( q \) is less than \( q_0 \) (a constant given parameter) then the ant selects the next job using Eq. (9), otherwise Eq. (10) is used for job selection.

IJMSEM email for subscription: info@msem.org.uk
1-Initialization
1-1-Set Parameters \((MaxIt, N \text{ ant}, \alpha, \beta, \tau_0, \rho, R_s)\)
1-2-Generating initial ants
2- Movement of each ant
2-1 Each ant selects the first job randomly and assign to earliest available machines
2-2 Other jobs select according to Eq.9 & Eq. 10 and assign to earliest available machines
2-3 If the ant ends its path
2-3-1 level updating is done according to Eq.11
2-4 Evaluate fitness function of each ant
3-If all ants complete their tour
3-1 best ant based on lower fitness function is selected
4-perform global updating based on Eq.12.
5-Stop if stopping criteria is met, otherwise go to step 2.

Fig. 12 Pseudo code of HACO

\[
s = \begin{cases} 
\arg \max_{s \in U} [\tau(r,u)]^d, |\eta(r,u)|^\beta, & \text{if } q \leq \eta_0 \\
R, & \text{otherwise}
\end{cases} \quad (9)
\]
\[
s = \begin{cases} 
\sum_{s \in U} [\tau(r,s)]^d, |\eta(r,s)|^\beta, & \text{if } s \in U \\
0, & \text{otherwise}
\end{cases} \quad (10)
\]
where \(r, s \) and \( U \) represent the job already selected, the next job to be selected by the ant and the unselected jobs respectively. \( P(r,s) \) represents the probability that job number \( 5 \), among the unselected jobs, is selected after job number \( r \).

(4) Local updating

Once the ants generate a solution, the pheromone level on the path is updated using a local update rule.

\[
\tau(i,j) = (1 - \rho)\tau(i,j) + \rho\tau_0,
\]
where \( 0 < \rho \leq 1 \), is a constant parameter.

(5) Global updating

After each ant has completed their tour and found the ant with the best solution, the global update rule is executed using Eq. (12)

\[
\tau(i,j) = (1 - \alpha)\tau(i,j) + \alpha\rho\Delta\tau(i,j),
\]
where

\[
\Delta\tau(i,j) = \begin{cases} 
(min(\sum_{s \in U} T_s))^{-1}, & \text{if } (i,j) \in \text{global-best-solution} \\
0, & \text{otherwise}
\end{cases}
\]

where \( 0 < \alpha \leq 1 \) is a constant parameter representing the pheromone evaporation.

(6) Termination test

If a termination condition occurs, the algorithm is stopped otherwise it is continued from Step 2. Fig. 12 shows the pseudo code for this algorithm.

3.2.2 Stochastic ant colony optimization (SACO)

This algorithm is similar to HACO expect for the movement mechanism of the Ant, i.e. Steps 2-1 and 2-2 presented in Fig. 11, which is described as follow: In a stochastic ant colony algorithm (SACO), the job sequence is created according to HACO. Then the jobs are assigned to the machines randomly.

3.3 Particle swarm optimization (PSO)

Particle Swarm Optimization (PSO) developed by Kennedy and Eberhart (1995) [18] and Kennedy and Eberhart (1997) [19] is inspired by the behavior of bird flocking and schooling fish. PSO is a population based stochastic optimization technique which has many similarities with evolutionary computation techniques. Pan et al. (2008) [22] applied DPSO algorithm to solve the no-wait flow shop scheduling problem with both makespan and total flow time criteria. In this paper we use two types of PSO namely Hybrid Particle Swarm Optimization (HPSO) and Stochastic Particle Swarm Optimization (SPSO). The difference between these two types of PSO is related to the type of job assignment to machines in both stages. In this study the job sequence in both stages is identical. The framework of HPSO and SPSO are described as follows:

3.3.1 Hybrid particle swarm optimization (HPSO)

Here the jobs are assigned to the machine at the earliest available time. The structure of HPSO is presented as follow:

(1) Initialize swarm

In HPSO, the potential solutions are named as particles which are spread randomly over the problem space. The number of particles is equal to the population size (PopSize). Each part of the particle structure is a random number between 0 and 1. As an example the structure of a solution with seven jobs and two machines in station 1 and four machines in station 2 is shown in Fig. 13.

Fig. 13 Structure of a solution for a seven-job problem using PSO

In this structure a job with a smaller number has higher priority. For example job 5 has the smallest number and hence it will be in the first priority. The jobs with subsequent priority are 4, 6, 7, 2, 3 and 1 respectively.

(2) Determination of the personal best and the global best

At the end of each iteration the particle best solution and the global best solution are determined. The best solution for each particle over the iterations to the presented iterations is determined as the particle best solution and the best solution among all particles over the iterations to the presented iterations is determined as the global best solution.

(3) Update of the Particle’s Position

Each particle moves according to a velocity based on three items namely; the distance between the previous position of the particle and the global best solution, the distance between the previous position of the particle and the particle best solution, and the current velocity. This definition is the PSO velocity concept.
Velocity of particle k at time \((t + 1)\) that is shown by 
\(V_{k}^{t+1}\) is computed by Eq. (13).
\[
V_{k}^{t+1} = w \times V_{k}^{t} + c_{1} \times r_{1} \times (P_{\text{best}} - P_{\text{current}}) + c_{2} \times r_{2} \times (G_{\text{best}} - P_{\text{current}}),
\]
where, \(V_{k}^{t}\) is the current velocity of particle \(k\) at time \(t\). Also \(w, c_{1}\) and \(c_{2}\) are the coefficients of the current velocity, the distance between the current position with \(P_{\text{best}}\) and the distance between current positions with \(G_{\text{best}}\), respectively. \(r_{1}\) and \(r_{2}\) are the constant parameters, and \(w\) is the parameter that decreases over the iterations. Also, \(r_{1}\) and \(r_{2}\) are the randomized parameters to create a randomized search. The particle position is updated at each iteration using Eq. (14).
\[
p_{k}^{t+1} = p_{k}^{t} + V_{k}^{t+1},
\]
where, \(p_{k}^{t+1}\) is the particle position of particle \(k\) at time \((t + 1)\) and \(p_{k}^{t}\) is the current position of particle \(k\).

### 4.1 Problem design

In this study, 36 test problems are used to evaluate the performance of the proposed algorithms. The data sets used in this research are created in various sizes in terms of the number of jobs, the number of machines in the first stage and second stage and the distribution of processing time in both stages. The problems are classified into two categories namely small and large scale problems. Tab. 1 demonstrates the number of jobs and the number of machines in a small and a large scale problem.

### 4.2 Parameter setting

In order to evaluate the performance of the proposed algorithms, some tuning is required to find the appropriate parameter value for the algorithms. For this purpose, experiments are performed on both small and large size problems. Tab. 2 displays the resulting tuned values obtained after initial simulation runs.

### 4.3 Experimental results

In this section, the results obtained using the proposed algorithms are presented and compared to each other. The performance measure considered is mean flow time. All algorithms were coded using MATLAB 2008a and run on a personal computer with a 2.53 GHz CPU and 2 GB memory. The performance of the algorithms was tested by solving 36 different problems (18 small scale problems and 18 large scale problems). Tab. 3 and Tab. 4 show the comparative results obtained using the six algorithms for small and large scale problems respectively.

Regarding the performance measures, a Relative Percent-Age Deviation (RPD) of mean flow time over the best solutions is used which is calculated as follows:
\[
\text{RPD} = \left( \frac{\text{Method}_{\text{sol}} - \text{Best}_{\text{sol}}}{\text{Best}_{\text{sol}}} \right) \times 100,
\]
where \(\text{Method}_{\text{sol}}\) is value and \(\text{Best}_{\text{sol}}\) is the best value for the mean flow time obtained from the algorithms. The results presented in Tab. 3 show that for small scale problems, in terms of mean flow time in 15 out of 18 cases, HICA outperforms the other algorithms. The results presented in Tab. 4 indicate that for large scale problems HICA outperforms the other algorithms in all instances. Furthermore Tab. 5 shows 95% confidence interval in terms of the RPD performance measure in both small and large scale problems. The results show that in both scales HICA obtained the lowest mean and deviation value compared with those of the other algorithms. In addition the results show that for small scale problems, the difference between the performance of SICA, HACO and HPSO are not significant. However, for the large scale problems after HICA, both HPSO and HACO significantly outperform the other algorithms. The results also indicate that CPU time of SPSO for all cases and in the both scales is the shortest. Furthermore, the results presented in Tab. 6 and Tab. 7 indicate the Tuckey test outcomes for small and large scales respectively.
The lower than 0.05 $P$ value, shown in these tables, indicates that the difference between the given algorithms is significant while the values higher than 0.05 indicate that the difference between the corresponding algorithms is not significant.

<table>
<thead>
<tr>
<th>No. jobs</th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>HACO</th>
<th>SACO</th>
<th>HICA</th>
<th>SICA</th>
<th>HPSO</th>
<th>SPSO</th>
<th>HACO</th>
<th>SACO</th>
<th>HICA</th>
<th>SICA</th>
<th>HPSO</th>
<th>SPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2</td>
<td>2</td>
<td>75.25</td>
<td>83</td>
<td>74.12</td>
<td>74.75</td>
<td>76.2</td>
<td>80.62</td>
<td>14.87</td>
<td>16.12</td>
<td>385.68</td>
<td>66.13</td>
<td>3.5</td>
<td>2.4</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>2</td>
<td>94</td>
<td>102.2</td>
<td>91.2</td>
<td>95.4</td>
<td>96.7</td>
<td>102.8</td>
<td>30.61</td>
<td>25.25</td>
<td>446.29</td>
<td>76.57</td>
<td>5.4</td>
<td>4.5</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>2</td>
<td>90.93</td>
<td>106.57</td>
<td>83.85</td>
<td>86.93</td>
<td>88.85</td>
<td>105.3</td>
<td>75.19</td>
<td>54.05</td>
<td>157.69</td>
<td>86.93</td>
<td>13.8</td>
<td>12.5</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>2</td>
<td>108</td>
<td>129.12</td>
<td>100.19</td>
<td>107</td>
<td>110.6</td>
<td>132.2</td>
<td>107.12</td>
<td>76.53</td>
<td>42.61</td>
<td>86.54</td>
<td>19.5</td>
<td>18.2</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>2</td>
<td>76.62</td>
<td>94.75</td>
<td>74.5</td>
<td>80.56</td>
<td>78.4</td>
<td>87.3</td>
<td>107.21</td>
<td>76.53</td>
<td>49.44</td>
<td>85.60</td>
<td>21.5</td>
<td>20.5</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>2</td>
<td>153.3</td>
<td>173</td>
<td>145.35</td>
<td>144.15</td>
<td>158.1</td>
<td>167.8</td>
<td>198.57</td>
<td>114.66</td>
<td>42.98</td>
<td>43.8</td>
<td>31.5</td>
<td>29.5</td>
</tr>
</tbody>
</table>

Table 3 Computational results for small scale problems

The sensitivity analysis of the algorithms versus number of jobs was performed and the results are presented in Fig. 15. The results show that in all cases HICA obtains better solutions compared with the other algorithms. In general the proposed HICA significantly outperforms other algorithms while SPSO yielded the lowest CPU time. Even though HICA CPU time is longer than that of SPSO, however, it was still very short in all cases studied. Therefore the proposed HICA can be considered an efficient algorithm for a no-wait two stage flexible flow shop with minimum flow time.

5 Conclusion

In this paper, six meta-heuristic algorithms were developed to solve the no-wait two stage flexible flow shop scheduling problem. A simulation model was developed to study the performance of the proposed algorithms in terms of mean flow time. For this purpose, numerical experiments were established to evaluate the performance of the proposed algorithms. In general the results revealed that the proposed HICA significantly outperforms other algorithms while SPSO yielded the lowest CPU time. Even though HICA CPU time is longer than that of SPSO, however, it was still very short in all cases studied. Therefore the proposed HICA can be considered an efficient algorithm for a no-wait two stage flexible flow shop with minimum flow time. For further research, it is recommended that the performance of the proposed algorithms with respect to other performance measures such as mean lateness and mean tar-
Table 4 Computational results for Large scale problems

<table>
<thead>
<tr>
<th>No. jobs</th>
<th>M_1</th>
<th>M_2</th>
<th>HACO</th>
<th>SACO</th>
<th>HICA</th>
<th>SICA</th>
<th>HPSO</th>
<th>SPSO</th>
<th>CPU time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>10</td>
<td></td>
<td>132.89</td>
<td>380.76</td>
<td>129.09</td>
<td>184.68</td>
<td>132.2</td>
<td>201.5</td>
<td>7869.82</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td></td>
<td>118.37</td>
<td>389.61</td>
<td>116.34</td>
<td>169.65</td>
<td>117.5</td>
<td>176.2</td>
<td>7676.37</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td></td>
<td>114.23</td>
<td>433.33</td>
<td>110.29</td>
<td>156.90</td>
<td>113.3</td>
<td>171.1</td>
<td>8354.96</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td></td>
<td>140.92</td>
<td>499.92</td>
<td>135.4</td>
<td>193.98</td>
<td>137.2</td>
<td>205.3</td>
<td>10585.76</td>
</tr>
<tr>
<td>80</td>
<td>10</td>
<td></td>
<td>125.89</td>
<td>461.94</td>
<td>122.85</td>
<td>180.62</td>
<td>123.3</td>
<td>186.4</td>
<td>11043.19</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td></td>
<td>118.37</td>
<td>509.15</td>
<td>116.65</td>
<td>170.66</td>
<td>116.4</td>
<td>176.4</td>
<td>11103.52</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td></td>
<td>158.19</td>
<td>565.81</td>
<td>151.44</td>
<td>225.58</td>
<td>154.8</td>
<td>245.8</td>
<td>14692.28</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td></td>
<td>143.78</td>
<td>541.40</td>
<td>137.41</td>
<td>209.74</td>
<td>137.9</td>
<td>216.8</td>
<td>14635.55</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td></td>
<td>134.48</td>
<td>538.40</td>
<td>128.72</td>
<td>194.01</td>
<td>131.4</td>
<td>199.7</td>
<td>14572.12</td>
</tr>
<tr>
<td>108</td>
<td>10</td>
<td></td>
<td>167.59</td>
<td>727.38</td>
<td>164.64</td>
<td>259.03</td>
<td>167.1</td>
<td>267.8</td>
<td>26548.21</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td></td>
<td>157.48</td>
<td>820.63</td>
<td>155.55</td>
<td>243.14</td>
<td>157.1</td>
<td>256.4</td>
<td>26957.25</td>
</tr>
<tr>
<td>120</td>
<td>10</td>
<td></td>
<td>207.45</td>
<td>905.21</td>
<td>202.2</td>
<td>894.51</td>
<td>204.1</td>
<td>336.9</td>
<td>35030.69</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td></td>
<td>185.58</td>
<td>852.48</td>
<td>179.9</td>
<td>289.55</td>
<td>183.1</td>
<td>299.6</td>
<td>36003.36</td>
</tr>
<tr>
<td>132</td>
<td>10</td>
<td></td>
<td>198.16</td>
<td>968.75</td>
<td>196.9</td>
<td>321.77</td>
<td>197.6</td>
<td>338.1</td>
<td>47610.73</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td></td>
<td>198.59</td>
<td>445.30</td>
<td>153.67</td>
<td>289.79</td>
<td>171.5</td>
<td>311.7</td>
<td>38773.92</td>
</tr>
<tr>
<td>Avg.</td>
<td></td>
<td></td>
<td>160.3</td>
<td>643.38</td>
<td>154.248</td>
<td>273.7</td>
<td>157.1</td>
<td>250.1</td>
<td>23430.3</td>
</tr>
</tbody>
</table>

Table 5 95% confidence interval of RPD for small and large scale problems

<table>
<thead>
<tr>
<th>Fitness function</th>
<th>Mean flow time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small scale</td>
<td>Large scale</td>
</tr>
<tr>
<td>HICA</td>
<td>(−0.044, 0.281)</td>
</tr>
<tr>
<td>SACO</td>
<td>(3.46, 8.92)</td>
</tr>
<tr>
<td>HPSO</td>
<td>(4.46, 7.84)</td>
</tr>
<tr>
<td>SPSO</td>
<td>(17.71, 26.12)</td>
</tr>
<tr>
<td>HACO</td>
<td>(3.302, 6.354)</td>
</tr>
<tr>
<td>SACO</td>
<td>(16.662, 26.343)</td>
</tr>
</tbody>
</table>

Table 6 Pair wise comparison of the algorithms in small scale

<table>
<thead>
<tr>
<th>Fitness function</th>
<th>Mean flow time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small scale</td>
<td>Large scale</td>
</tr>
<tr>
<td>HICA</td>
<td>SACO</td>
</tr>
<tr>
<td>SICA</td>
<td>HPSO</td>
</tr>
<tr>
<td>HPSO</td>
<td>SPSO</td>
</tr>
<tr>
<td>SPSO</td>
<td>HACO</td>
</tr>
<tr>
<td>SACO</td>
<td></td>
</tr>
</tbody>
</table>

Table 7 Pair wise comparison of the algorithms in small scale

<table>
<thead>
<tr>
<th>Fitness function</th>
<th>Mean flow time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small scale</td>
<td>Large scale</td>
</tr>
<tr>
<td>HICA</td>
<td>SACO</td>
</tr>
<tr>
<td>SICA</td>
<td>HPSO</td>
</tr>
<tr>
<td>HPSO</td>
<td>SPSO</td>
</tr>
<tr>
<td>SPSO</td>
<td>HACO</td>
</tr>
<tr>
<td>SACO</td>
<td></td>
</tr>
</tbody>
</table>

References


