A bi-objective scheduling problem on batch machines via a Pareto-based ant colony system

Rui Xu, Huaping Chen, Xueping Li

1. Introduction

Nowadays microprocessors, memory chips, and other semiconductor related devices are a part of daily life, appearing in electronics ranging from personal computers to cellular phones. The semiconductor industry has already become one of the largest industries in the world, and semiconductor manufacturers need to utilize their resources effectively to confront the huge demand and severe competition in the marketplace. Furthermore, there is increasing pressure for semiconductor manufacturers to improve their overall performance, such as machine utilization and on-time delivery. Effective scheduling is one of the most economical and executable potential solutions to such challenges.

Batch processing is a very common procedure in semiconductor manufacturing for the avoidance of setups and/or facilitation of material handling. A batch is defined as a group of jobs that have been processed jointly (Brucker et al., 1998) and a batch scheduling problem consists of grouping jobs on each machine into batches which are scheduled either in serial (called s-batching) or in parallel (called p-batching). On an s-batching machine, the processing time of a batch equals the sum of the processing times of all jobs that form the batch. On a p-batching machine, several jobs can be processed as a batch simultaneously, and the length of a batch is the largest processing time of its jobs.

In semiconductor manufacturing, p-batching is a much more important issue, occurring in many processes such as oxidation/deposition/diffusion of wafer fabs and the burn-in operation in test facilities (Moench et al., 2011). For instance, the burn-in operation of the final testing phase of semiconductor manufacturing tests the integrated circuits for defects by subjecting them to thermal stress for an extended period of time. In this way, latent defects may be discovered that would otherwise only be found in the operating environment. While very important, the operations' lengthy processing time compared with the other operations in semiconductor manufacturing creates a bottleneck in the final testing phase. Consequently efficient scheduling is of great concern to overall performance.

In addition, p-batching is commonly observed in other modern industrial environments, such as in chemical, food and mineral processing, pharmaceutical, and metalworking industries as well as with wafer and environmental stress screening chamber fabrication.

This paper focuses on p-batching scheduling, which is also known as the scheduling of a batch-processing machine (BPM) or
batch machine in the literature (Mathirajan and Sivakumar, 2006; Moench et al., 2011). On a BPM, a fixed maximum batch size exists as the capacity of the batch machine, and a batch machine is capable of accommodating a group of jobs simultaneously as long as the sum of the job sizes in the batch is less than or equal to the machine capacity. The processing time of each batch is determined by the longest job processing time among the jobs contained in the batch. Finally, once a batch is being processed, it is non-preemptable during the long period of processing.

In this research, we consider the BPM problem of scheduling on identical batch machines arranged in parallel with dynamic job arrivals and non-identical job sizes to minimizing makespan ($C_{\text{max}}$) and maximum tardiness ($T_{\text{max}}$) simultaneously. Note that the target problem is complicated by the considerations of different constraints and multiple objectives. First, a machine environment involving multiple batch machines instead of one requires an additional decision about machine assignment to obtain a final solution. Secondly, dynamic job arrivals and non-identical job sizes, which reflect realistic conditions in most manufacturing systems, are considered. Lastly, two performance measures of $C_{\text{max}}$ and $T_{\text{max}}$, which are among the most commonly used criteria in the batch scheduling research, are considered. Makespan is a measure of system utilization while maximum tardiness is a measure of performance in meeting customer due dates. This target problem can be denoted by $\min \{P_{\text{parallel}}(S_j, P_{\text{batch}}(C_{\text{max}}, T_{\text{max}})\}$ and the details of the problem statement and formulation will be given in Section 3.

To the best of our knowledge, the BPM scheduling problem for optimizing multi-objectives has not been fully investigated in the literature. Only Kashan et al. (2010) studied a NP-hard scheduling problem on a BPM while optimizing the same objectives considered in the target problem. However, they do not consider the constraints of dynamic job arrivals and multiple BPMs, which often arises from real-world production systems and occurs in a more general environment. Besides, this then leads to the problem becoming much more complicated and being NP-hard as well. In view of the complexity of the target problem, we extend the framework of ant colony system, which has been proven effective for solving complex single objective optimization problems, to explore its potential for solving multi-objective scheduling problems.

The rest of this paper is outlined as follows: Section 2 provides a literature review, including batch machine scheduling problems with different constraints. Section 3 illustrates the problem under study and presents the bi-objective integrated mathematical model for the problem. Section 4 describes the proposed multi-objective ant colony optimization algorithm, which employs several distinctive features based on the problem-specific knowledge. Then, taking into account different test instances and evaluation metrics, we conduct experimental studies as presented in Section 5. The paper concludes in Section 6.

2. Related work

In recent years, a body of research has started to address the scheduling problem of a BPM. Initially, researchers started from the simplest model—scheduling for a single BPM. Ikura and Gimple (1986) were probably the first authors study the BPM problem. They proposed an $O(n^2)$ algorithm for a single BPM problem with identical job processing times, unit job sizes, dynamic job arrivals, and an objective of minimizing makespan. Lee et al. (1992) first presented a detailed description for burn-in operation and then studied a problem in which the processing time of jobs were identical and the relationship between release time ($r_i$) and due date ($d_i$) was acceptable, i.e., $r_i \leq r_j \leq d_j$, implying $d_i \leq s_j$.

They considered minimizing maximum lateness, $L_{\text{max}}$, and the number of tardy jobs, $\sum U_i$, using dynamic programming algorithms. Uzsoy and Yang (1997) considered the problem of minimizing total weighted completion time, $\sum w_i C_i$, and provided several heuristics and a branch-and-bound algorithm. Jolai (2005) showed that the problem of minimizing the number of tardy jobs, $\sum U_i$, is NP-hard. He developed a dynamic programming algorithm with polynomial time complexity for a fixed number of job families and limited BPM capacity.

Real-world production systems usually require BPMs arranged in parallel in order to prevent the system from being blocked by the unavailability (e.g., breakdown) of a single BPM. Accordingly, parallel systems have been the focus on several studies. Lee et al. (1992) examined the worst-case error bound of any list-scheduling algorithm for minimizing $C_{\text{max}}$ with unit job sizes and used the longest processing time (LPT) rule to the case of identical BPMs. Chang et al. (2004) proposed a simulated annealing (SA) algorithm to minimize $C_{\text{max}}$ on BPMs in parallel with non-identical job sizes and compared the results obtained from the CPLEX Solver. Kashan et al. (2008) developed a hybrid genetic algorithm (GA) to minimize makespan when scheduling different size jobs. A simulated annealing approach was employed as a comparator algorithm. Damodaran and Velez-Gallego (2010) proposed a constructive heuristic to schedule jobs with non-identical job sizes and arbitrary release times and compared the proposed algorithm with several heuristics in the literature with respect to solution quality and computational cost. Wang and Chou (2010) provided a mixed integer programming model for identical BPMs with non-identical job sizes and dynamic job arrivals. They also proposed two meta-heuristics based on SA and GA, which were combined with a dynamic programming algorithm.

To make a BPM scheduling problem closer to the real-world situation, the constraints of non-identical job sizes and dynamic job arrivals should be considered. Considering jobs with non-identical sizes, Uzsoy (1994) presented complexity results for $C_{\text{max}}$ and $L_{\text{max}}$ criteria. He also provided several heuristics and a branch-and-bound algorithm. To schedule a BPM to minimize $C_{\text{max}}$ with different job sizes, Melouk et al. (2004) and Damodaran et al. (2006) proposed two different algorithms: a simulated annealing approach and a genetic algorithm, respectively. Kashan et al. (2006) proposed two different genetic algorithms based on different encoding schemes to minimize makespan with non-identical job sizes. Kashan and Karimi (2008) developed an ant colony framework in two versions, varying according to the type of embedded heuristic information, to minimize total weighted completion time of a single BPM with incompatible job families and arbitrary job sizes. Considering jobs with arbitrary release times, Lee and Uzsoy (1999) first presented polynomial and pseudo polynomial-time algorithms for several special cases and developed various heuristics to minimize makespan on a single BPM in the presence of dynamic job arrivals. Sung and Choung (2000) proposed some heuristics to minimize $C_{\text{max}}$ on a single burn-in oven in semiconductor manufacturing with different job release times. Wang and Uzsoy (2002) combined a dynamic programming algorithm with a random key-based representation genetic algorithm to minimize $L_{\text{max}}$ on a single BPM in the presence of job release times. Malve and Uzsoy (2007) considered the problem of minimizing maximum lateness $L_{\text{max}}$ on identical BPMs with dynamic job arrivals. They developed a family of iterative improvement heuristics and combined them with a genetic algorithm.

Clearly, although BPM scheduling with different constraints has been extensively researched, these studies have largely been limited to BPM problems with only a single objective function (e.g., $C_{\text{max}}$), and optimizing a single objective generally may lead to deterioration of other possible objectives. In semiconductor manufacturing...
environments, multiple objectives are more realistic and thus need to be considered in optimization of the overall system performance. Of the limited work on multi-objective BPM problems available in respected literature, Li et al. (2009) addressed a scheduling problem on BPMs arranged in parallel with incompatible job families, dynamic job arrivals, sequence-dependent setup times, and quality run requirements of advanced process control. They proposed an ant colony optimization algorithm to minimize the total weighted tardiness and makespan simultaneously, as long as the total size of jobs in the batch does not exceed machine capacity denoted by \( b \). The schedules are constructed. We could obtain the values of \( C_{\text{bm}} \), the processing time of batch \( b \) on machine \( m \) in \( \sigma \) is denoted by \( C_{\text{bm}}(\sigma) \), i.e., \( C_{\text{bm}} = S_{\text{bm}} + P_{\text{bm}} \) and \( S_{\text{bm}} = \text{Max}(C_{\text{bm-1m}}, R_{\text{bm}}) \). The tardiness of batch \( b \) on machine \( m \) in \( \sigma \) is denoted by \( T_{\text{bm}}(\sigma) \), i.e., \( T_{\text{bm}} = \text{Max}(0, C_{\text{bm}} - \text{Min}(d_j j \in b \text{bm})) \). Consequently, the objectives \( C_{\text{max}} \) and \( T_{\text{max}} \) of \( \sigma \) are the maximum completion time and tardiness among all batches in solution \( \sigma \), i.e., \( C_{\text{max}} = \text{Max}(C_{\text{bm}}(\sigma) b \in \sigma \) and \( T_{\text{max}} = \text{Max}(T_{\text{bm}}(\sigma) b \in \sigma \).

We used Fig. 1 as an example to illustrate the scheduling of BPMs arranged in parallel with non-identical job sizes and dynamic job arrivals. In Fig. 1, there were nine jobs to be scheduled on two BPMs, each with a machine capacity of 7. At first, the nine jobs were grouped into five batches under the capacity constraint. The values of \( P_{\text{bm}} \) and \( R_{\text{bm}} \) could be only calculated depending on the jobs assigned in each batch as shown in Fig. 1. Then, the five batches were scheduled on two BPMs separately. Machine M1 was assigned to process three batches (B1, B3, B5) and machine M2 was assigned to process the other two batches (B2, B4). After sequencing the formed batches on each machine, a feasible solution \( \sigma \) was constructed. We could obtain the values of \( C_{\text{bm}} \) and \( T_{\text{bm}} \) by considering not only the jobs assigned in each batch but also the batch sequences on each machine. Finally, the \( C_{\text{max}} \) value of solution \( \sigma \) was the maximum batch completion time, i.e., \( C_{\text{max}} = \text{Max}(9, 10, 12, 20, 21) = 21 \), and the \( T_{\text{max}} \) value of solution \( \sigma \) was the maximum batch tardiness, i.e., \( T_{\text{max}} = \text{Max}(2, 1, 5, 0) = 5 \).

As can be seen, a feasible solution is usually achieved by making three decisions: batch formation, machine assignment, and batch sequencing. Every time when a BPM is available to process another batch, more than one combination of jobs to form the next batch may be possible. On one hand, forming a batch with more jobs inside reduces the waste of machine capacity. On the other hand, forming a batch with less jobs inside may reduce the processing time and tardiness of each job in the batch.

### 3. Problem statement

The focused problem \( P_m | p_j s_j, p-batch | C_{\text{max}}, T_{\text{max}} \) in this paper can be stated as follows. Assume there are \( n \) compatible jobs to be grouped into several batches (the number of batches is uncertain) and scheduled on \( p \) identical BPMs arranged in parallel. Each job \( j \) is associated with a processing time \( p_j \), a due date \( d_j \), a release time \( r_j \), and a corresponding size \( s_j \). We assume that all the data is deterministic and known a priori. Each BPM \( m \) has a maximum machine capacity denoted by \( B \) and is able to process multiple jobs simultaneously in a batch at a time, as long as the total size of all jobs in the batch does not exceed \( B \). The schedules are non-preemptive, i.e., once processing of a batch on a machine is initiated, no job can be added or removed from the batch until the processing of the batch is complete.

The final purpose of scheduling is to obtain a feasible solution \( \sigma \) by grouping \( n \) jobs into several batches and scheduling the formed batches on \( p \) BPMs so that the bi-criteria of makespan and maximum tardiness is optimized. Specifically, given a non-empty batch \( b \), the processing time of batch \( b \) is determined by the longest processing time among all jobs in batch \( b \), i.e., \( P_b = \text{Max}(p_j j \in b) \).

Similarly, the release time of batch \( b \) is determined by the latest release time among all jobs in batch \( b \), i.e., \( R_b = \text{Max}(r_j j \in b) \). These two variables are independent from the batch scheduling process, i.e., \( P_{\text{bm}} = P_b \) and \( R_{\text{bm}} = R_b \). However, the completion time and tardiness of batch \( b \) cannot be obtained until all batches are assigned and sequenced on BPMs (i.e., a feasible solution \( \sigma \) is obtained). That is to say, the calculation of completion time and tardiness depends not only on how to form batches but also how to schedule them. After a feasible solution \( \sigma \) is constructed, the completion time of batch \( b \) on machine \( m \) in \( \sigma \) is denoted by \( C_{\text{bm}}(\sigma) \), i.e., \( C_{\text{bm}} = S_{\text{bm}} + P_{\text{bm}} \) and \( S_{\text{bm}} = \text{Max}(C_{\text{bm-1m}}, R_{\text{bm}}) \). The tardiness of batch \( b \) on machine \( m \) in \( \sigma \) is denoted by \( T_{\text{bm}}(\sigma) \), i.e., \( T_{\text{bm}} = \text{Max}(0, C_{\text{bm}} - \text{Min}(d_j j \in b \text{bm})) \).

![Fig. 1](image-url)

**Fig. 1.** An illustration of a feasible solution for scheduling on BPMs in parallel with non-identical job sizes and dynamic job arrivals. Nine jobs are grouped into five batches and then assigned to two BPMs. The machine capacity is set to seven. The makespan and maximum tardiness of the solution are 21 and 5, respectively.

---

Please cite this article as: Xu, R., et al., A bi-objective scheduling problem on batch machines via a Pareto-based ant colony system. International Journal of Production Economics (2013), [http://dx.doi.org/10.1016/j.ijpe.2013.04.053](http://dx.doi.org/10.1016/j.ijpe.2013.04.053)
other hand, waiting for jobs to form a full batch may delay the jobs that have already arrived. Hence, a compromise between increasing the utilization of BPMs and decreasing the delay time of waiting jobs is required. In the above example, batch B4 was formed with the two jobs J1 and J7. Note that J1 arrived at time 3, versus J7 at time 11. In order to accommodate the upcoming job J7 in batch B4 and increase BPM utilization, J1 was delayed to be processed together with J7. Consequently, there was an idle time of machine M2 while waiting for job J7 between the processes of B2 and B4. Similarly, there was another two idle times before the process of B1 and B2, separately. Machine assignment aims at distributing batches suitably to machines so that workload among BPMs is balanced. Finally, on each BPM, the formed batches must be processed in a proper sequence in order to optimize the performance measures. In the example above, although B3 had arrived at time 3, it was sequenced after B1 on machine M1 and hence could not be processed until the process of B1 was completed at time 9.

4. Proposed Pareto based ant colony system

4.1. Overview

Ant Colony Optimization (ACO) is a constructive population based meta-heuristic search algorithm for optimization problems. Over the past few years, ACO has been extended to deal with multi-objective optimization problems, known as multi-objective ant colony optimization (MOACO). To date, MOACO has been applied to a large number of combinatorial optimization problems such as aluminum casting (Gravel et al., 2002), portfolio selection (Doerner et al., 2004), resource allocation (Chaharsooghi and Kermani, 2008), vehicle routing (Fuellerer et al., 2009), production and maintenance scheduling (Berrichi et al., 2010), and flow shop scheduling (Yagmahan and Yenisey, 2010) problems. For more details on MOACO, we refer readers to the surveys by Garcia-Martinez et al. (2007) and Angus and Woodward (2009).

From a given initial point, ants start walking based on the decision rule of choosing the next job to be visited. This movement is guided by the pheromone trails and the heuristic information associated with the problem under study. After constructing a complete solution, applying a local search procedure is possible to improve the obtained solution. When all of the ants have generated their solutions, the Pareto set is updated, keeping all non-dominated solutions generated up to that point. Then pheromone trails are updated by considering the quality of the candidate solutions generated as well as a certain level of pheromone evaporation. Each algorithm presents different ways to choose nodes and update the pheromone trails.

In the literature one can find various alternatives for the implementation of ACO algorithms for solving multi-objective combinatorial optimization problems. They usually differ from each other with respect to the single objective ACO algorithms they are based on, such as Ant System (AS), Ant Colony System (ACS), and Max-Min Ant System (MMAS), and vary in their algorithmic components, including, for example, different number of pheromone trails or heuristic information (Garcia-Martinez et al., 2007). In our case, based on the performance of ACS for single objective scheduling problems (Ying and Liao, 2004; Huang and Yang, 2008; Zhan et al., 2010) and the characteristics of the problem under study, the ACS framework is extended to multi-objective optimization by considering multiple matrices for both pheromone trails and heuristic information, with each matrix corresponding to one objective. Furthermore, the proposed algorithm is Pareto-based so that a set of non-dominated solutions will be returned as output. Thus we call the proposed algorithm a Pareto-based Ant Colony System (PACS) algorithm.

In order to make the PACS algorithm more efficient and effective for solving the target problem, we employ three distinctive features using problem-specific knowledge as follows.

1. A new solution construction mechanism is proposed which encodes the three decisions directly as a complete solution to explore the entire search space.
2. A candidate list strategy is incorporated into the procedure of the solution construction which guarantees that the search will concentrate on the regions containing high-quality solutions.
3. A form of dynamic heuristic information is adopted which allows for a more precise guidance of the search process for each objective.

First, based on the constructive characteristics of PACS, we employ a new solution construction mechanism. It considers the three decisions simultaneously in order to encode a feasible solution directly, which ensures PACS has the ability to explore the entire solution space. Although PACS is a promising optimization approach, searching in such a huge solution space effectively remains challenging. The other two features are designed to deal with this challenge. As for the second feature, we introduce a candidate list strategy considering different constraints to reduce the search space and to guide the search toward promising regions in the solution space. By using the candidate list strategy, we can obtain better performance in a limited computational time with the same iteration. For the third feature, a form of dynamic heuristic information is adopted based on the problem-specific knowledge of each objective. Each objective is transformed into a corresponding space-related metric. Based on the three distinctive features, the underlying optimization ability of PACS can be utilized more efficiently. Details of the proposed algorithm are described in the following subsections.

4.2. Solution construction mechanism

As mentioned above, solving the target problem is usually achieved by making three decisions, which may result in a huge solution space. Although meta-heuristics including ACO are known as promising optimization approaches, searching in large solution spaces effectively remains challenging (Chiang et al., 2010). Consequently, in order to reduce the solution search space, the three decisions are often addressed independently in the literature. The most common way is to employ a meta-heuristic such as a GA to optimize the batch formation decision first, then use dispatching rules to schedule the formed batches on BPMs, also called the ‘batch first, schedule second’ strategy in the literature (Damodaran and Vélez-Gallego, 2012).

Although this strategy is useful for reducing the search space, appealing solutions may be excluded from the search space and unreachable by the meta-heuristic (Chiang et al., 2010). Note that when a job is ready before a machine is available, the release time of this job should no longer be a valid constraint. However, the ‘batch first, scheduled second’ strategy does not take this fact into account, possibly hurt solution quality by considering the release time constraint unnecessarily. Alternatively, based on the constructive characteristics of ACO, we propose a new solution construction mechanism, which can encode the three decisions directly as a complete solution as shown in Fig. 2. In the new solution construction mechanism, each ant selects the first available machine (chosen randomly if all machines are available) as the current machine $m$, forming a batch $b$ based on current machine $m$‘s available time $t$ and assigning it to the current
machine. Details of the solution construction mechanism of PACS will be illustrated in Section 4.4.

Fig. 3 illustrates how a complete solution was constructed with the new solution construction mechanism in PACS. This example uses the same instance with nine jobs to be scheduled on two BPMs given in Section 3. Initially, since all machines were free, ant \(a\) selected \(M_1\) as the current machine \(m\) and opened a new batch \(B_1\) on it. The current machine \(m\)’s available time was zero and set as the current time \(t\). Since \(J_2\) was the only available job before time \(t\), it was assigned in \(B_1\) when it was empty. By using the decision rule presented in Section 4.2.6, \(J_5\) was selected from the candidate jobs into \(B_1\) when it was non-empty. When \(B_1\) was full, it was closed and scheduled on the current machine \(M_1\). Afterwards, \(M_2\) was chosen as the current machine \(m\) since it was the first one available between the two BPMs. Batch formation and machine assignment continued until all jobs were scheduled.

From the above example, it can be seen that batch formation is the most crucial and non-trivial of the three decisions. Thus, the proposed PACS algorithm mainly focuses on searching for good batch formation; the other two decisions are determined implicitly during the procedure of solution construction. Furthermore, the three decisions are correlative and dependent upon each other. For example, the machine assignment decision is determined based on batches that have been assigned to each BPM. Meanwhile, the batch formation decision is made according to the current machine \(m\)’s available time.

4.2.1. Pheromone trails

The quantity of a pheromone laying on a component represents the past experience of ants with respect to choosing this component. The definition of the pheromone trails is crucial, and a poor choice at this stage of the algorithm design will result in poor performance.

In the case of our problem, since a feasible solution is constructed by choosing a job from a candidate list into the current batch iteratively, the value \(\tau_{kmu}\) must refer to the desirability of assigning the next job \(u\) from the candidate list into the current batch \(b\) on machine \(m\). However, the current batch varies along with the assigned jobs in the batch. In order to obtain the value \(\tau_{kmu}\), we define the pheromone trails \(r_{ij}\) as the desirability of having job \(i\) in a same batch with job \(j\) as in the work (Xu et al., 2012) and the value

<table>
<thead>
<tr>
<th>Step</th>
<th>Unscheduled jobs</th>
<th>Candidate jobs</th>
<th>machine batch</th>
<th>job</th>
<th>Cmax</th>
<th>Tmax</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{2,4,5,6,1,3,7,8,9}</td>
<td>{2}</td>
<td>M1 B1</td>
<td>J2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>{4,5,6,1,3,7,8,9}</td>
<td>{1,5,6,7,8,9}</td>
<td>M1 B1</td>
<td>J5</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>{4,6,1,3,7,8,9}</td>
<td>{4}</td>
<td>M2 B2</td>
<td>J4</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>{6,1,3,7,8,9}</td>
<td>{6,8}</td>
<td>M2 B2</td>
<td>J6</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>{1,3,7,8,9}</td>
<td>{3}</td>
<td>M1 B3</td>
<td>J3</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>{1,7,8,9}</td>
<td>{1}</td>
<td>M2 B4</td>
<td>J1</td>
<td>19</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>{7,8,9}</td>
<td>{7,8,9}</td>
<td>M2 B4</td>
<td>J7</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>{8,9}</td>
<td>{9}</td>
<td>M1 B5</td>
<td>J9</td>
<td>21</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>{8}</td>
<td>{8}</td>
<td>M1 B5</td>
<td>J8</td>
<td>21</td>
<td>5</td>
</tr>
</tbody>
</table>

Fig. 3. An example to illustrate the solution construction mechanism of the PACS algorithm. Nine jobs are grouped in five batches and then processed on two BPMs. The upper subfigure describes all the steps of the solution construction and the lower subfigure depicts the route of ant \(a\) constructing a feasible solution for the instance.

Please cite this article as: Xu, R., et al., A bi-objective scheduling problem on batch machines via a Pareto-based ant colony system. International Journal of Production Economics (2013), http://dx.doi.org/10.1016/j.ijpe.2013.04.053
\( s_{bm,u}^k \) can be obtained indirectly by the following equation.

\[
s_{bm,u}^k = \begin{cases} \frac{\sum_{b \in \bm} r_{bm}^k}{|\bm|} & \text{if } \bm \neq \emptyset \\ 1 & \text{otherwise} \end{cases}
\]  

(1)

where \( r_{bm}^k \) is the average of the pheromone values between the job \( u \) to be added and the jobs already in the current batch; if batch \( b \) is empty, then \( r_{bm}^k \) is set to one. For each objective \( k \), the pheromone trails are stored in a matrix with the number of elements corresponding to the number of jobs.

4.2.2. Heuristic information

In general, static and dynamic heuristic information are the main types of heuristic information used by ACO algorithms. In the static case, the heuristic values are computed once at initialization time and then remain unchanged throughout the whole algorithm’s run. In the dynamic case, the heuristic information depends on the partial solution constructed thus far and therefore has to be computed at each step of an ant’s walk (Dorigo and Stützle, 2004).

In our case, heuristic information is used to guide the search during batch formation. Since the situation of an ongoing batch varies according to the assigned jobs in the batch and the batches scheduled on the machine, static heuristic information may not be an appropriate option, especially when considering that job release time may no longer be a valid constraint when a job is ready before a machine is available. Consequently, dynamic heuristic information is constructed based on the problem-specific knowledge of both objectives, each of which is transformed into a corresponding space-related metric as follows.

4.2.3. Heuristic information for \( C_{\text{max}} \)

In this subsection, we concentrate on the construction of the heuristic information for the objective of makespan. Although the machine environment involves several BPMs instead of a single one, the optimal objective of makespan and the other constraints are the same. Hence we can construct the heuristic information with respect to \( C_{\text{max}} \) based on the waste and idle space (\( \text{WIS} \)) concept proposed by Xu et al. (2012) as well. The following presents an extensional definition of \( \text{WIS} \) for this work.

**Definition 1.** Suppose there exists a feasible schedule \( \sigma \) and \( b \) is an arbitrary batch on machine \( m \) of schedule \( \sigma \). \( \text{WIS}_{bm} \) denotes the waste and idle space of batch \( b \) on machine \( m \), which can be calculated by Eq. (2). \( \text{WIS}_m \) expresses the waste and idle space of machine \( m \), which equals the sum of \( \text{WIS}_{bm} \) on machine \( m \). \( \text{WIS}(\sigma) \) signifies the waste and idle space of schedule \( \sigma \), which equals the maximum \( \text{WIS}_m \) among all BPMs.

We use Fig. 4 to illustrate the concept of \( \text{WIS}_{bm} \) graphically. As can be seen, the value of the \( \text{WIS}_{bm} \) can be calculated with the following expression:

\[
\text{WIS}_{bm} = B \cdot (C_{bm} - C_{(b-1)m}) - \sum_{j \in \bm} s_j \cdot p_j
\]  

(2)

where the first expression on right side of Eq. (2) presents the whole space including the waste and idle space and the job processing space; the second expression denotes the space occupied by the jobs in the batch.

Based on the definition of \( \text{WIS} \), Xu et al. (2012) proved that solving \( \min \{r_j, s_j, B\} \) is equivalent to minimizing the value of \( \text{WIS} \) of solution \( \sigma \). In other words, for a fixed number of jobs to be scheduled on machine \( m \), the value of the \( \text{WIS}_{bm} \) is proportional to the machine completion time. Therefore, decreasing the \( \text{WIS}_m \) for each machine may produce a better solution. Since the \( \text{WIS}_m \) is the sum of the \( \text{WIS}_{bm} \) and the value of the \( \text{WIS}_{bm} \) is fixed once batch \( b \) is scheduled on machine \( m \), decreasing the \( \text{WIS}_{bm} \) of machine \( m \) means decreasing the \( \text{WIS}_{bm} \) of the current batch \( b \) on machine \( m \).

Based on the above analysis, we produce the reduced value \( \Delta \text{WIS}_u \) of a candidate job \( u \) by Eq. (3)

\[
\Delta \text{WIS}_u = \text{WIS}_{bm} - \text{WIS}_{bm} = s_u \cdot p_u \cdot B \cdot (S_{bm} - S_{bm}) - (P_{bm} - P_{bm})
\]

(3)

where \( \text{WIS}_{bm} \) denotes the waste and idle space of batch \( b \) on machine \( m \) after job \( u \) has been assigned into the current batch. However, a candidate job may increase or decrease the \( \text{WIS}_{bm} \) depending on the relationship between the chosen job and the current batch. That is to say, the reduced value \( \Delta \text{WIS}_u \) may be positive or negative. Considering this fact and the restriction that heuristic information has to have a positive value, we propose the heuristic information with respect to \( C_{\text{max}} \) as follows:

\[
\eta_{bm,u} = \begin{cases} 1 + \frac{1}{\Delta \text{WIS}_{bm}} & \text{if } \Delta \text{WIS}_{bm} > 0 \\ \frac{1}{s_u \cdot p_u \cdot B \cdot (S_{bm} - S_{bm}) - (P_{bm} - P_{bm})} + 1 & \text{if } \Delta \text{WIS}_{bm} \leq 0 \end{cases}
\]

(4)

where

\[
\Delta \text{WIS}_{\text{max}} = \max \{\Delta \text{WIS}_u | u \in \mathcal{N}_{bm}\}
\]

and

\[
P_{bm} = \max \{p_u, P_{bm}\}
\]

and

\[
S_{bm} = \max \{r_u, S_{bm}\}
\]

and

\[
S_{bm} = \max \{r_u, S_{bm}\}
\]

Xu et al. (2012) considered a single BPM problem and scheduled batches after all the batches were formed. Hence, \( C_{(b-1)m} \) could not be obtained until a batch sequence was finished. Note that due to the different solution construction mechanism in our approach, we can obtain \( C_{(b-1)m} \) during the solution construction procedure, and thus it is much more accurate for calculating heuristic information. Furthermore, the value of \( \eta_{bm,u} \) is limited to the interval of \((0, 2]\).
4.2.4. Heuristic information for $T_{\text{max}}$

In order to construct heuristic information with regard to $T_{\text{max}}$, we first propose a corresponding space-related concept of tardiness space ($TS$) as follows:

**Definition 2.** Suppose there exists a feasible schedule $\alpha$ and $b$ is an arbitrary batch on machine $m$ of schedule $\alpha$. $TS_{bm}$ denotes the tardiness space of batch $b$ on machine $m$, which can be calculated by Eq. (5). $TS(\alpha)$ signifies the tardiness space of schedule $\alpha$, which equals to the maximum $TS_{bm}$ among all batches on BPsMs.

$$TS_{bm} = B \cdot T_{bm} = B \cdot \max(C_{bm} - D_{bm}, 0)$$  \hspace{1cm} (5)

We use Fig. 5 to illustrate the concept of $TS_{bm}$ graphically. As can be seen, the value of the $TS_{bm}$ can be calculated by Eq. (5), which equals to the maximum tardiness of batch $b$ on machine $m$ multiplied by the capacity of machine $B$.

From Definition 2, we know that the $TS(\alpha)$ is the maximum $TS_{bm}$ and it is obvious that the $TS(\alpha)$ is proportional to the objective of $T_{\text{max}}$ for schedule $\alpha$. In other words, the objective of minimizing the $T_{\text{max}}$ can be transformed into minimizing the $TS(\alpha)$ of schedule $\alpha$. From this point of view, the heuristic information with regard to $T_{\text{max}}$ is needed for batching jobs together to minimize the increase in the $TS(\alpha)$ as much as possible. Because the $TS(\alpha)$ is the maximum tardiness space among all batches, considering the solution construction mechanism, the heuristic information for decreasing the tardiness space $TS_{bm}$ of the current batch should be constructed to the extent possible.

**Proposition 1.** Let $b$ be the current batch on machine $m$ with at least one job, $u$ be a job to be assigned, and $b'$ be the new batch into which job $u$ has been assigned. If job $u$ satisfies the capacity constraint, then the new tardiness space $TS_{bm'}$ must be larger or equal to the original tardiness space $TS_{bm}$.

**Proof.** From Eq. (5), we know $TS_{bm} = B \cdot \max(C_{bm} - D_{bm}, 0)$. After job $u$ has been assigned into batch $b$, the new tardiness space $TS_{bm'} = B \cdot \max(C_{bm'} - D_{bm'}, 0)$. Let $\Delta TS_{u}$ denote the difference between the $TS_{bm}$ and $TS_{bm'}$, and we can produce the $\Delta TS_{u}$ with the following expression:

$$\Delta TS_{u} = TS_{bm'} - TS_{bm} = B \cdot [(C_{bm} - C_{bm'}) + (D_{bm} - D_{bm'})]$$  \hspace{1cm} (6)

Since $C_{bm} = S_{bm} + P_{bm}$, we can transform the above equation into as follows:

$$\Delta TS_{u} = B \cdot [(S_{bm} - S_{bm'}) + (P_{bm} - P_{bm'}) + (D_{bm} - D_{bm'})]$$  \hspace{1cm} (7)

From the definitions of batch processing time and batch tardiness, i.e., $P_{bm} = \max(p_j | j \in \text{bm})$ and $D_{bm} = \min(d_j | j \in \text{bm})$, we can obtain the new batch processing time $P_{bm'}$ and new batch tardiness $D_{bm'}$ after job $u$ has been assigned into. Obviously, the values of $P_{bm'}$ and $D_{bm'}$ are determined by the longest job processing time and the smallest job tardiness, respectively. Consequently, both the terms of $(P_{bm} - P_{bm'})$ and $(D_{bm} - D_{bm'})$ are larger or equal to zero.

From the description of batch starting time, i.e., $S_{bm} = \max(R_{bm}, C_{b-1:m})$, we can calculate the new batch starting time $S_{bm'}$ after job $u$ has been assigned into considering two different cases as follows:

For the case of $R_{bm} \geq C_{b-1:m}$, $S_{bm'} = \max(r_u, R_{bm})$; Otherwise, $S_{bm'} = \max(r_u, C_{b-1:m})$.

As can be seen, the value of $S_{bm'}$ is determined by both the latest job release time and the previous batch completion time. To sum up the two cases above, the term of $(S_{bm} - S_{bm'})$ is larger or equal to zero as well.

Since all the three terms on right side of Eq. (7) are larger or equal to zero, we know that the $\Delta TS_{u}$ must be a non-negative value, which means that the $TS_{bm'}$ must be larger or equal than the $TS_{bm}$.

Based on Proposition 1, we know that the tardiness space value of $TS_{bm}$ must not decrease after adding a new job. It is natural to think that we should select a candidate job with a smaller $\Delta TS_{u}$.

However, due to the new solution construction mechanism, we have to take the relationship between a candidate job and the current batch into account to decide the preferred $\Delta TS_{u}$. Considering this fact, the heuristic information for $T_{\text{max}}$ is constructed as follows:

$$\eta_{bm,u} = \left\{ \begin{array}{ll}
\frac{A}{\Delta TS_{bm} + B \cdot (S_{bm} - S_{bm'}) + (P_{bm} - P_{bm'}) + (D_{bm} - D_{bm'}) + 1} & \text{if } r_u \leq S_{bm} \\
\frac{1}{B \cdot (S_{bm} - S_{bm'}) + (P_{bm} - P_{bm'}) + (D_{bm} - D_{bm'}) + 1} & \text{if } r_u > S_{bm} 
\end{array} \right. \hspace{1cm} (8)
$$

where

$$\Delta TS_{max} = \max(\Delta TS_{u} | u \in \mathcal{N}_{bm})$$

and

$$D_{bm} = \min(d_u, D_{bm})$$

and

$$P_{bm} = \max(p_u, P_{bm})$$

and

$$S_{bm'} = \max(r_u, S_{bm'})$$

Note that there are two cases for the heuristic information $\eta_{bm,u}$ depending on whether job $u$ has arrived or not before the batch starting time $S_{bm}$. For the first case of $r_u \leq S_{bm}$, the selection of the next job $u$ with the larger $\Delta TS_{u}$ value is preferred. In this case, the positive value of $\Delta TS_{u}$ results mainly from the smaller amount of time for due date of the current batch following the addition of the next job $u$. Considering this point, a larger value of $\Delta TS_{u}$ may be produced if job $u$ is assigned into a later batching process. Hence, a job $u$ with a larger $\Delta TS_{u}$ has more possibility to be selected for this case. For the other case of $r_u > S_{bm}$, the selection of a next job $u$ with a smaller $\Delta TS_{u}$ value is preferred. In this case, the positive value of $\Delta TS_{u}$ is mainly due to an increase in the length of the
starting time of the current batch after adding the next job \( u \). Based on this point, the greater the \( \Delta TS_u \), the lower the desirability of having job \( u \) in batch \( b \). Hence, a job \( u \) with a larger \( \Delta TS_u \) has less possibility of being selected for this case. Furthermore, the value of \( \eta_{bm}^2 \) is limited to the interval of \((0, 2]\) as well.

4.2.5. Candidate list

In general, a candidate list contains a number of the best rated choices to be considered at each construction step according to some heuristic criterion. Considering our case, the size constraint is intuitively the only mandatory constraint. That is to say, the size of the candidate job must not exceed the residual machine capacity as shown in Eq. (9). Otherwise, there are no other mandatory constraints that exist.

\[ N_{bm} = \{ u | s_u \leq (B - \sum_{j \in bm} s_j) \} \tag{9} \]

According to our solution construction mechanism, an important decision point needs to be considered: the time \( t \) when a machine becomes available to form a batch. At this point, suppose there exists an upcoming job \( u \) satisfying the size constraint in Eq. (9). A decision has to be made as to whether to open a new batch to accommodate the upcoming job or to wait for the upcoming job to form a full batch. On the one hand, we intend to form a batch with more jobs inside so that machine capacity is not wasted. On the other hand, waiting for an upcoming job to form a full batch may delay the jobs that have already arrived. Hence, this decision is not an easy one.

In Section 4.2.3, we have known that \( WIS_u \) is an overall indicator which considers three indicators (job size, job processing time, and job release time) together. Moreover, the search space that does not increase the \( WIS_u \) has a greater possibility of generating a better solution than other spaces. Consequently, the \( WIS_u \) indicator is used as the other candidate condition to make a compromise between increasing utilization of BPMs and decreasing delay time of waiting jobs. An upcoming job \( u \) is regarded as a candidate job if it satisfies the constraint of \( \Delta WIS_u \leq 0 \) as shown in Eq. (10). Otherwise, it is excluded from the candidate list. To sum up, the final candidate list for our problem is presented as follows:

\[ N_{bm} = \{ u | s_u \leq (B - \sum_{j \in bm} s_j) \} \]

\[ S_u \leq (B - \sum_{j \in bm} s_j) \tag{10} \]

where

\[ P_{bm} = \max(p_u, P_{bm}) \]

and

\[ S_{bm} = \max(s_u, S_{bm}) \]

Note that, although the procedure to find the candidate jobs is dynamic, no more computational cost is required to calculate the \( \Delta WIS_u \) values since they have already been obtained during the calculation of heuristic information.

4.2.6. Decision rule

Given the pheromone trails, the heuristic information, and the candidate list as described above, a candidate job \( u \) is selected to be assigned into the current batch \( b \) according to a pseudorandom-proportional rule that can be stated as follows:

\[ u = \left\{ \begin{array}{ll}
\arg \max_{i \in N_{bm}} \left\{ \sum_{k=1}^{K} \left( \frac{P_k \cdot \tau_{bm}^k}{\tau_{bm}^k} \right) \cdot \left( \sum_{k=1}^{K} \left( P_k \cdot \eta_{bm}^k \right)^2 \right) \right\} & \text{if } q \leq q_0 \\
\hat{u} & \text{otherwise}
\end{array} \right. \]

where \( q \) is a random number uniformly distributed in \([0,1]\) and \( q_0 \) is a parameter (\( 0 \leq q_0 < 1 \)) to be set by the user to represent the probability that a job is chosen, thus providing the highest aggregate value of pheromone and heuristic information. The random variable \( \hat{u} \) is selected according to the probability distribution given

\[ \hat{u} = \left\{ \begin{array}{ll}
\sum_{k=1}^{K} \left( P_k \cdot \tau_{bm}^k \right) \cdot \left( \sum_{k=1}^{K} \left( P_k \cdot \eta_{bm}^k \right)^2 \right) & \text{if } u \in N_{bm} \\
0 & \text{otherwise}
\end{array} \right. \tag{12} \]

The probability distribution is biased by the parameter \( \gamma \), which determines the relative influence of the pheromone trails and the heuristic information. The probability of choosing a job outside \( N_{bm} \) is zero. \( P_k \) denotes the objective weight for each objective \( k \), which is a random number uniformly distributed in \([0,1]\) for our case. Note that, it also can be determined by a decision maker in order to reflect the user’s preference.

4.3. Update of pheromone trails

There are two pheromone updating processes in this PACS algorithm. A local pheromone update rule is performed to the pheromone matrix for each objective \( k \) once an ant has added a job to a non-empty batch during the solution construction. When an ant selects a job \( u \), the amount of pheromone on the elements \( \tau_{k,j} \) is decreased for each objective \( k \). The local pheromone update rule for these elements can be stated as follows:

\[ \tau_{k,j}(c + 1) = (1 - \rho) \cdot \tau_{k,j}(c) + \rho \cdot \tau_{k,j0} \tag{13} \]

where \( \tau_{k,j0} \) is the initial value of trails and \( \rho \in (0, 1) \) is a parameter to control the speed of pheromone evaporation.

Global pheromone trails are updated once the colony has computed a set of solutions, and every non-dominated solution (belonging to the Pareto set) is rewarded. Let \( S_p \) be this set of non-dominated solutions. The quantity of a pheromone deposited on a solution component for the \( k \)th pheromone structure is defined as follows:

\[ \Delta \tau_{k,j}(c) = \left\{ \begin{array}{ll}
1 + \frac{1}{|S_p| + 1} & \text{if } \sigma \in S_p, \sigma \neq i \\&\ j \\
0 & \text{otherwise}
\end{array} \right. \tag{15} \]

\[ \Delta \tau_{k,i} = \sum_{j \in S_p} \Delta \tau_{k,i} \]

where \( \Delta \tau_{k,i} \) is the value obtained for objective \( k \), which is proportional to the solution quality. It is important to note that in PACS, the pheromone trails update, both evaporation and new pheromone deposit, only applies to the non-dominated solutions of \( S_p \) rather than all solutions. In this way, the computational complexity of the pheromone update at each iteration is reduced significantly. Since the initial pheromone value \( \tau_{k,j0} \) is set to one for each objective, we add one to the deposited pheromone \( \Delta \tau_{k,j} \), which could limit the pheromone values \( \tau_{k,j} \) to the interval \([\tau_{k,j0}, \Delta \tau_{k,j}]\) for each iteration.

4.4. Description of the proposed PACS algorithm

Based on the rules and the concepts presented in the previous subsections, we provide a detailed description of the PACS algorithm as shown in the following flow chart, which briefly explains each step of the developed algorithm.
Algorithm 1. The PACS algorithm.

1. Set PACS parameters \( N_c \) (number of iterations), \( N_a \) (number of ants);
2. Initialize pheromone \( \tau^k_0 \) for each objective \( k \);
3. Initialize \( c = 1 \); /* \( c \) is the iteration counter; */
4. while \( c \leq N_c \) do
5.   for \( a \leftarrow 1 \) to \( N_a \) do
6.     \( \bigcup^a(c) = \{ J_1, \ldots, J_n \} \) /* \( \bigcup^a(c) \) is the unscheduled job set for each ant */
7.   end
8. while \( \bigcup^a(c) \neq \emptyset \) do
9.   for \( a \leftarrow 1 \) to \( N_a \) do
10.  \( N_{\text{bm}}^a(c) = \{ u | s_u \leq (B - \sum_{j \in \text{bm}} s_j) \text{ and } s_u p_u > B \cdot (s_{bm'} - s_{bm}) + B \cdot (P_{bm'} - P_{bm}) \} \)
11.  /* Construct the candidate job set for the current batch */
12.  if \( N_{\text{bm}}^a(c) = \emptyset \) then
13.    \( t^a(c) \leftarrow \min \{ C_m \} \) /* Set the earliest available time as the current time */
14.    \( B_{\text{bm}}^a(c) \leftarrow \emptyset \) /* Establish an empty batch based on the current time */
15.    \( A_{\text{bm}}^a(c) = \{ u | r_u \leq t^a(c) \} \) /* Construct the available job set at the current time */
16.  end
17.  else
18.    \( B_{\text{bm}}^a(c) \leftarrow J_u, J_u \in \{ u | d_u = \min \{ d_u \} \text{ and } J_u \in A_{\text{bm}}^a(c) \} \); /* Select the job \( J_u \) with the smallest due date from \( A_{\text{bm}}^a(c) \) to the current batch */
19.  end
20. end
21. end
22. else
23.    \( B_{\text{bm}}^a(c) \leftarrow J_u, J_u \in N_{\text{bm}}^a(c) ; \) /* Select the next job \( J_u \) from the candidate job set to the current batch by equation (11) and (12) */
24. end
25. \( \bigcup^a(c) \leftarrow \bigcup^a(c) \setminus J_u ; \) /* Remove the selected job \( J_u \) from the unscheduled job set */
26. \( \tau^a_{i,j}(c + 1) = (1 - \rho) \cdot \tau^a_{i,j}(c) + \rho \cdot \Delta \tau^a_{i,j}(c) \) /* Local pheromone trails updating */
27. end
28. end
29. \( X^* \leftarrow \sigma, \sigma \in PF(X^*) \) /* Store the Pareto front of the \( c^\text{th} \) generation to the Pareto archive */
30. \( \tau^a_{i,j}(c + 1) = (1 - \rho) \cdot \tau^a_{i,j}(c) + \rho \cdot \Delta \tau^a_{i,j}(c) \) /* Global pheromone trails updating */
31. end
32. \( X \leftarrow \sigma, \sigma \in PF(X^*) \) /* Return the Pareto front of \( X^* \) */

5. Computational experiments

5.1. Experimental setup

For testing effectiveness of the proposed algorithm, random test problem instances were generated based on Xu et al. (2012). To cover various types of problems, we considered several levels of influencing inputs. First, we generated random problem instances for \( n = 20, 50, 100, \) and \( 200 \) jobs, respectively. After specifying the number of jobs in a given instance, an integer processing time \( p_j \) was generated from the uniform distribution \([8, 48]\) for each job. Also, we considered small-sized, large-sized, and normal-sized problem instances, which were presented by an integer job size \( s_j \) from the uniform distributions \([1, 15]\), \([15, 35]\), and \([1, 40]\), respectively. The number of machines was assumed to be 4, and the maximum capacity of each BPM was set to be 40 for all instances.

We generated the release time \( r_j \) as \( r_j = U(0, \alpha) \cdot C_{\text{max}}^{LB} \), where \( U(a, b) \) denoted a continuous uniform distribution between \( a \) and \( b \). Note that the term \( C_{\text{max}}^{LB} \) was calculated based on the lower bound proposed by Damodaran and Velez-Gallego (2010). Here \( \alpha \) was a user defined parameter allowing us to control the distribution of job arrivals. After appointing the release times and processing times of jobs, we generated the due date \( d_j \) as \( d_j = r_j + U(1, \beta) \cdot p_j \). Here \( \beta \) was another parameter to control the distribution of due date tightness. Note that if \( \alpha = 0 \), all jobs are available at time zero. As \( \alpha \) increases, the jobs arrive over a longer interval. If \( \beta = 1 \), the due date of each job is equal to its release...
time plus its processing time, meaning that all jobs must begin processing at the moment they arrive. As \( \beta \) increases, the difference between release times and due dates becomes larger. As a result, the scheduling problems become less constrained and more easily solved.

For each combination of the input values of \( \alpha, \beta, n, \) and \( s_j \), five instances were generated for each problem and consequently there were a total of \( 72 \times 5 = 360 \) problem instances. For each test problem, all the algorithms were coded in C++ and independently executed 30 times on a computer with Intel Xeon \( \times 5335 \) 2.00 GHz CPU and 2 GB RAM. The factors, their levels, and their ranges are shown in Table 1. Each category of problems was represented with a run code. For example, a problem with 20 jobs and 4 machines, job sizes generated from the discrete uniform distribution within \([1,15]\), job release times generated with \( \alpha = 0.3 \), and job due dates generated with \( \beta = 3 \) was shown by J1S1R1D1.

In order to evaluate the efficacy and strength of the PACS algorithm, we first compared the PACS algorithm with a comparator algorithm, called a sequence based multi-objective genetic algorithm (SMOGA), which was proposed based on the framework of the non-dominated sorting genetic algorithm II (NSGA-II) by Kashan et al. (2010). The SMOGA algorithm constructed a feasible solution using the ‘batch first, schedule second’ strategy. First, each chromosome was defined as a sequence of jobs, the parameterized uniform crossover and the swap mutation were employed as the genetic operators. Then the first-fit dispatching rule was applied to form batches. For a detailed description of SMOGA, see Kashan et al. (2010). The SMOGA algorithm was designed to solve a multi-objective scheduling problem on a single BPM, hence we extended the algorithm to solve the multi-objective scheduling problem on BPMs in this paper by using a well-known earliest release time (ERT) heuristic to make machine assignment decision. Furthermore, in order to validate if the proposed PACS algorithm was effective by the parameters adopted in OPACS were set in the same way as J1S1R1D1.

Since the proposed PACS algorithm was extended based on the framework of ACS, we incorporated the default parameters used in the literature and conducted several pilot experiments to determine the relatively important parameters in this paper. For the number of ants \( m \), we executed the PACS algorithm using the parameter setting of \( m = \{10, 20, 50, 100, 200\} \) and left all other choices the same. In fact, we obtained a better trade-off between the solution quality and computational cost when using a fixed number of ants. Thus, we set \( m = 20 \) for all problem instances. To determine the proper value of parameter \( \gamma \), the performance of the PACS algorithm was evaluated by using values 1–10. Through our comparison with different settings, as we found 2–4 appropriate for most instances, the value of 3 was selected. Considering the deposited pheromone amount in the pheromone update procedure, the initial pheromone values were set to one for each objective so that the pheromone values at each iteration could be limited to the interval \( [0, \Delta \rho] \). Tuning the pheromone evaporation rate \( \rho \) and the parameter \( q_0 \) could allow regulation of the degree of exploration and exploitation. In the pilot experiments, we verified the performance of PACS using the parameter setting of \( \rho \in [0.02, 0.1, 0.5, 0.95] \) and \( q_0 \in [0.1, 0.4, 0.9] \). The results showed that PACS could obtain the relatively best performance for most instances when \( \rho = 0.1 \) and \( q_0 = 0.4 \). The max-iterations of \( \rho \) was set to 100 as the termination rule for the PACS algorithm. The final parameter values selected for the proposed PACS algorithm are presented in Table 2. We set the same key parameters for the two MOACO algorithms in such a way that the advantage of using the distinctive features can be illustrated more convincingly. Otherwise, the parameters for SMOGA were set by default values from Kashan et al. (2010).

### 5.2. Evaluation metrics

The performance of a multi-objective optimization algorithm is usually evaluated from two aspects (Zitzler et al., 2003). First, the obtained non-dominated set should be as close to the Pareto front as possible. That is to say, the distance of the resulting non-dominated front to the Pareto front should be minimized. Second, the solutions in the obtained non-dominated set should be distributed as diversely as possible. However, the desired distribution of the Pareto front might differ depending on the problem being solved. In this paper, due to the characteristics of solutions, the search space is highly discrete, which leads to a non-uniform distribution of the non-dominated solutions. As a result, a more appropriate choice is to evaluate the distance of the obtained non-dominated solutions.

![Fig. 6](https://example.com/figure6.png) An illustration of the concept of HV. \( A \) is a Pareto set with four non-dominated solutions. \( f^* \) is a reference point. The black area denotes the HV of \( A \).

### Table 1

Factors and levels.

<table>
<thead>
<tr>
<th>Factors</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>R1 = 0.3, R2 = 0.5, R3 = 0.7</td>
</tr>
<tr>
<td>( \beta )</td>
<td>D1 = 3, D2 = 5</td>
</tr>
<tr>
<td>Number of jobs ( n )</td>
<td>J1 = 20, J2 = 50, J3 = 100, J4 = 200</td>
</tr>
<tr>
<td>Number of machines ( m )</td>
<td>m = 4</td>
</tr>
<tr>
<td>Job size ( s_j )</td>
<td>S1 = Uniform [1,15], S2 = Uniform [15,35], S3 = Uniform [1,40]</td>
</tr>
<tr>
<td>Job processing time ( \beta )</td>
<td>Uniform [8,48]</td>
</tr>
<tr>
<td>Machine capacity ( B )</td>
<td>40</td>
</tr>
</tbody>
</table>

### Table 2

Parameter settings.

<table>
<thead>
<tr>
<th>( m )</th>
<th>20</th>
<th>( q_0 )</th>
<th>0.4</th>
<th>( \gamma )</th>
<th>3</th>
<th>( \rho )</th>
<th>0.1</th>
<th>( N_c )</th>
<th>100</th>
<th>( \epsilon )</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 )</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>( f_2 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Please cite this article as: Xu, R., et al., A bi-objective scheduling problem on batch machines via a Pareto-based ant colony system. International Journal of Production Economics (2013), http://dx.doi.org/10.1016/j.ijpe.2013.04.053
dominated set from the Pareto front. In this paper, the following two metrics were employed.

(1) Distance from reference set ($I_D$): This metric was proposed by Czyzak and Jaszkiewicz (1998). It can be stated as follows:

$$I_D(A) = \frac{\sum_{x \in R} [\min_{y \in A} (d(x, y))]}{|R|}$$

Let $R$ be the reference solution set. Given a Pareto set $A$, $I_D(A)$ provides information about the average distance from a solution in the reference set $R$ to the closest solution in $A$. A smaller value of $I_D(A)$ indicates that $A$ is closer to $R$. If the reference set $R$ is defined as a set of Pareto optimal solutions whose objective vectors are uniformly distributed on the Pareto front, $I_D$ will indicate the closeness of the set $A$ to the Pareto front and the distribution of the solutions in $A$. However, obtaining the Pareto front for a multi-objective BPM instance is difficult. Moreover, even the Pareto optimal solutions themselves may be distributed non-uniformly. Alternatively, the non-dominated solutions obtained by all three algorithms in 30 runs on a instance were combined, and those solutions which remained non-dominated were used as the reference set in our experiment.

(2) Hypervolume ($I_H$): This metric was suggested by Zitzler et al. (2001) to indicate the area in the objective space that is dominated by at least one solution of the non-dominated set. In practise, it can be calculated as follows:

$$I_H(A) = \int_{v \in \mathbb{R}^n} \ldots \int_{v \in \mathbb{R}^m} 1 \cdot dv$$

where $HV(f(x), f^*) = \int f^*(f^1, \ldots, f^m) \ldots \int h_0(x)$ is the Cartesian product of the closed intervals $[f_i(x), f_i^*], i = 1, \ldots, m$. Here we assume the objectives are to be minimized and the reference point $f^* = (f^*1, \ldots, f^*m)$ is the ideal worst point, i.e., $f_i^* = \max_{x \in \mathcal{X}} f_i(x), i = 1, \ldots, m$. An example with two objectives is given in Fig. 6, where the objective vectors of the solutions in the set $A$ are $a, b, c,$ and $d$. The area in black indicates the hypervolume (HV) of $A$.

It is obvious if solution $x_1$ dominates solution $x_2$, then $HV(f(x_1), f^*) \geq HV(f(x_2), f^*)$. Hence, $I_H$ reflects the closeness of the non-dominated set to the Pareto front. Larger the $I_H$, the closer

### Table 3

<table>
<thead>
<tr>
<th>Run code</th>
<th>SMOGA</th>
<th>OPACS</th>
<th>PACS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$I_D$</td>
<td>$I_H$</td>
<td>Time</td>
</tr>
<tr>
<td>J1S1R1D1</td>
<td>0.5429</td>
<td>0.1924</td>
<td>8.38</td>
</tr>
<tr>
<td>J1S1R1D2</td>
<td>0.5862</td>
<td>0.2877</td>
<td>7.18</td>
</tr>
<tr>
<td>J1S1R2D1</td>
<td>0.4862</td>
<td>0.3829</td>
<td>6.76</td>
</tr>
<tr>
<td>J1S1R2D2</td>
<td>0.4541</td>
<td>0.1865</td>
<td>7.05</td>
</tr>
<tr>
<td>J1S1R3D1</td>
<td>0.5550</td>
<td>0.2339</td>
<td>7.27</td>
</tr>
<tr>
<td>J1S1R3D2</td>
<td>0.4119</td>
<td>0.1487</td>
<td>12.21</td>
</tr>
<tr>
<td>J1S2R1D1</td>
<td>0.3360</td>
<td>0.1721</td>
<td>12.05</td>
</tr>
<tr>
<td>J1S2R1D2</td>
<td>0.3450</td>
<td>0.1501</td>
<td>12.66</td>
</tr>
<tr>
<td>J1S2R2D1</td>
<td>0.3884</td>
<td>0.4065</td>
<td>11.60</td>
</tr>
<tr>
<td>J1S2R2D2</td>
<td>0.1258</td>
<td>0.8776</td>
<td>12.42</td>
</tr>
<tr>
<td>J1S2R3D1</td>
<td>0.4679</td>
<td>0.5279</td>
<td>12.42</td>
</tr>
<tr>
<td>J1S2R3D2</td>
<td>0.5285</td>
<td>0.1851</td>
<td>9.17</td>
</tr>
<tr>
<td>J1S3R1D1</td>
<td>0.5377</td>
<td>0.1787</td>
<td>9.98</td>
</tr>
<tr>
<td>J1S3R1D2</td>
<td>0.4020</td>
<td>0.4129</td>
<td>10.55</td>
</tr>
<tr>
<td>J1S3R2D1</td>
<td>0.3417</td>
<td>0.5171</td>
<td>11.76</td>
</tr>
<tr>
<td>J1S3R2D2</td>
<td>0.3870</td>
<td>0.5197</td>
<td>10.04</td>
</tr>
<tr>
<td>J1S3R3D2</td>
<td>0.3147</td>
<td>0.6406</td>
<td>11.83</td>
</tr>
</tbody>
</table>

### Table 4

<table>
<thead>
<tr>
<th>Run code</th>
<th>SMOGA</th>
<th>OPACS</th>
<th>PACS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$I_D$</td>
<td>$I_H$</td>
<td>Time</td>
</tr>
<tr>
<td>J2S1R1D1</td>
<td>0.7705</td>
<td>0.0436</td>
<td>11.92</td>
</tr>
<tr>
<td>J2S1R1D2</td>
<td>0.8322</td>
<td>0.0250</td>
<td>11.00</td>
</tr>
<tr>
<td>J2S1R2D1</td>
<td>0.7388</td>
<td>0.0526</td>
<td>11.90</td>
</tr>
<tr>
<td>J2S1R2D2</td>
<td>0.8180</td>
<td>0.0429</td>
<td>12.01</td>
</tr>
<tr>
<td>J2S1R3D1</td>
<td>0.8259</td>
<td>0.0490</td>
<td>11.84</td>
</tr>
<tr>
<td>J2S1R3D2</td>
<td>0.6770</td>
<td>0.1017</td>
<td>10.48</td>
</tr>
<tr>
<td>J2S2R1D1</td>
<td>0.6506</td>
<td>0.1110</td>
<td>22.26</td>
</tr>
<tr>
<td>J2S2R1D2</td>
<td>0.6447</td>
<td>0.0772</td>
<td>20.77</td>
</tr>
<tr>
<td>J2S2R2D1</td>
<td>0.4904</td>
<td>0.2857</td>
<td>22.16</td>
</tr>
<tr>
<td>J2S2R2D2</td>
<td>0.5333</td>
<td>0.2232</td>
<td>22.35</td>
</tr>
<tr>
<td>J2S2R3D1</td>
<td>0.4567</td>
<td>0.5589</td>
<td>22.42</td>
</tr>
<tr>
<td>J2S2R3D2</td>
<td>0.4730</td>
<td>0.4145</td>
<td>22.95</td>
</tr>
<tr>
<td>J2S3R1D1</td>
<td>0.6791</td>
<td>0.0924</td>
<td>21.59</td>
</tr>
<tr>
<td>J2S3R1D2</td>
<td>0.7284</td>
<td>0.0822</td>
<td>19.26</td>
</tr>
<tr>
<td>J2S3R2D1</td>
<td>0.6346</td>
<td>0.1449</td>
<td>20.44</td>
</tr>
<tr>
<td>J2S3R2D2</td>
<td>0.7278</td>
<td>0.0711</td>
<td>15.92</td>
</tr>
<tr>
<td>J2S3R3D1</td>
<td>0.7510</td>
<td>0.1867</td>
<td>18.65</td>
</tr>
<tr>
<td>J2S3R3D2</td>
<td>0.6641</td>
<td>0.2473</td>
<td>18.59</td>
</tr>
</tbody>
</table>

Please cite this article as: Xu, R., et al., A bi-objective scheduling problem on batch machines via a Pareto-based ant colony system. International Journal of Production Economics (2013), http://dx.doi.org/10.1016/j.ijpe.2013.04.053
the corresponding non-dominated set is to the Pareto front. In addition, from the fact that for two sets $A_1$ and $A_2$, $A_1 \supseteq A_2 \iff \forall (a_1, a_2) \in \mathcal{A}_1 \times \mathcal{A}_2$, it is deduced that a larger $l_q$ implies that the non-dominated set covers the Pareto front more completely. Furthermore, $l_q$ is the only unary measure which is consistent with the Pareto dominance relationship, i.e., if a set dominates another one, it always has a better $l_q$ (Mei et al., 2011). For this reason, $l_q$ is nowadays one of the most commonly used measures for evaluating multi-objective optimization algorithms.

In this study, all the metrics were computed based on the normalized objective vectors of the non-dominated solutions, which were obtained by

$$f_i = \left( \frac{f_i - f_{i_{\text{min}}}}{f_{i_{\text{max}}} - f_{i_{\text{min}}}} \right), \quad i = 1, 2$$

where $f_1$ and $f_2$ denote the makespan and maximum tardiness, respectively, $f_{i_{\text{max}}}$ and $f_{i_{\text{min}}}$ stand for the maximal and minimal values of $f_i$ among all the results obtained over the 30 runs of the three compared algorithms. Since the elements of the normalized objective vectors always lie in the interval $[0,1]$, the point $(1,1)$ was adopted as the reference point in our following experiment.

5.3. Experimental results

A computational study was conducted to evaluate the efficiency and effectiveness of the proposed algorithm. For each problem we randomly generated five instances and provided the average values over five instances in tables.

Tables 3–6 present the average value of $l_p$, $l_q$ and run time over the ten independent runs of the compared algorithm on the various instances, respectively. Column 1 represents the run code for the instances. Columns 2 and 3 report the $l_p$ and $l_q$ values of SMOGA, respectively, where each instance is tested ten times. Note that $l_p$ is to be minimized while $l_q$ is to be maximized. Column 4 shows the mean run time of SMOGA for ten run times. Similarly, Columns 5–10 report the performance metrics and run time for two MOACO algorithms with ten run times respectively. For each problem and each metric, the result that performs better than others is in boldface (with smallest $l_p$, while with greatest $l_q$).

### Table 5

<table>
<thead>
<tr>
<th>Run code</th>
<th>SMOGA</th>
<th>OPACS</th>
<th>PACS</th>
</tr>
</thead>
<tbody>
<tr>
<td>J3S1R1D1</td>
<td>0.1239</td>
<td>0.0894</td>
<td>41.35</td>
</tr>
<tr>
<td>J3S1R1D2</td>
<td>0.1345</td>
<td>0.0894</td>
<td>41.92</td>
</tr>
<tr>
<td>J3S2R1D1</td>
<td>0.1478</td>
<td>0.2847</td>
<td>45.46</td>
</tr>
<tr>
<td>J3S2R1D2</td>
<td>0.1523</td>
<td>0.2966</td>
<td>45.55</td>
</tr>
<tr>
<td>J3S3R1D1</td>
<td>0.1678</td>
<td>0.3971</td>
<td>46.25</td>
</tr>
<tr>
<td>J3S3R1D2</td>
<td>0.1723</td>
<td>0.3971</td>
<td>46.35</td>
</tr>
<tr>
<td>J3S1R2D1</td>
<td>0.1878</td>
<td>0.4928</td>
<td>47.15</td>
</tr>
<tr>
<td>J3S1R2D2</td>
<td>0.1923</td>
<td>0.4928</td>
<td>47.25</td>
</tr>
<tr>
<td>J3S2R2D1</td>
<td>0.2078</td>
<td>0.5985</td>
<td>48.05</td>
</tr>
<tr>
<td>J3S2R2D2</td>
<td>0.2123</td>
<td>0.5985</td>
<td>48.15</td>
</tr>
<tr>
<td>J3S3R2D1</td>
<td>0.2278</td>
<td>0.6942</td>
<td>49.05</td>
</tr>
<tr>
<td>J3S3R2D2</td>
<td>0.2323</td>
<td>0.6942</td>
<td>49.15</td>
</tr>
</tbody>
</table>

### Table 6

<table>
<thead>
<tr>
<th>Run code</th>
<th>SMOGA</th>
<th>OPACS</th>
<th>PACS</th>
</tr>
</thead>
<tbody>
<tr>
<td>J4S1R1D1</td>
<td>0.0917</td>
<td>0.0798</td>
<td>30.85</td>
</tr>
<tr>
<td>J4S1R1D2</td>
<td>0.1045</td>
<td>0.0798</td>
<td>30.95</td>
</tr>
<tr>
<td>J4S1R2D1</td>
<td>0.1173</td>
<td>0.0871</td>
<td>31.85</td>
</tr>
<tr>
<td>J4S1R2D2</td>
<td>0.1301</td>
<td>0.0871</td>
<td>32.95</td>
</tr>
<tr>
<td>J4S2R1D1</td>
<td>0.1438</td>
<td>0.1926</td>
<td>33.85</td>
</tr>
<tr>
<td>J4S2R1D2</td>
<td>0.1565</td>
<td>0.1926</td>
<td>34.95</td>
</tr>
<tr>
<td>J4S3R1D1</td>
<td>0.1702</td>
<td>0.2955</td>
<td>35.85</td>
</tr>
<tr>
<td>J4S3R1D2</td>
<td>0.1829</td>
<td>0.2955</td>
<td>36.95</td>
</tr>
<tr>
<td>J4S1R2D1</td>
<td>0.1956</td>
<td>0.3983</td>
<td>37.85</td>
</tr>
<tr>
<td>J4S1R2D2</td>
<td>0.2083</td>
<td>0.3983</td>
<td>38.95</td>
</tr>
<tr>
<td>J4S2R2D1</td>
<td>0.2210</td>
<td>0.4911</td>
<td>39.85</td>
</tr>
<tr>
<td>J4S2R2D2</td>
<td>0.2337</td>
<td>0.4911</td>
<td>40.95</td>
</tr>
</tbody>
</table>

Please cite this article as: Xu, R., et al., A bi-objective scheduling problem on batch machines via a Pareto-based ant colony system. International Journal of Production Economics (2013), http://dx.doi.org/10.1016/j.ijpe.2013.04.053
Overall, the table results clearly show that PACS performs better than OPACS and SMOGA for both performance metrics in all categories of test problem instances, especially for the problems with a large number of jobs. With respect to the trend for increasing reported performance values, we expect PACS to outperform the other two algorithms as the number of jobs increases. However, there is no significant difference among the performance of three algorithms for small problem cases. This is due to the small size of the feasible solution space, for which all algorithms could obtain good performances.

Compared to the SMOGA algorithm, PACS performed better on 66 out of the total 72 problems for the $I_D$ metric and on 67 out of the total 72 problems for the $I_H$ metric. The SMOGA algorithm was superior on 6 $J_1$ instances for the $I_D$ metric and on 5 $J_1$ instances for the $I_H$ metric. It can be observed that SMOGA outperformed PACS only on the problem instances with 20 jobs. This is because PACS is better able to explore a larger solution space and has a chance to find better solutions as the number of jobs increases. However, as SMOGA can only optimize one of the three decisions, the other two decisions are determined by simple heuristics. Thus, it may not be able to explore the search space thoroughly, possibly excluding good solutions from the search space.

Compared to the OPACS algorithm, PACS performed better on 59 out of the total 72 problems for the $I_D$ metric and on 61 out of the total 72 problems for the $I_H$ metric. The OPACS algorithm was superior on 9 $J_1$ instances, 2 $J_2$ instances, and 2 $J_3$ instances for the $I_D$ metric and on 11 $J_1$ instances for the $I_H$ metric. It can be seen that most of the problems that performed better for OPACS were generated for 20 job instances. This is due to that PACS employs three distinctive features that enhance the optimization ability of the original algorithm. First, PACS incorporates a new solution construction mechanism which encodes the three decisions directly as a complete solution, allowing more possibility to explore the entire search space. However, exploring the entire solution space is a great challenge for PACS as well. Consequently, a candidate list strategy is devised to guarantee the search of regions containing high-quality solutions. In addition, based on the problem-specific knowledge for each objective, a form of dynamic heuristic information is adopted to give a more precise guidance of the search process. These three features establish the superior performance of PACS as compared to the OPACS algorithm.

The average performance values and computation time with respect to different problem characteristics are summarized in Table 7. As the number of jobs increases, the number of batches formed becomes larger generally. In this condition, there are more decisions to be made and consequently a large solution space to be explored. As can be seen, the reported $I_D$ values become lower, and $I_H$ values become higher when the number of jobs increases, which demonstrates the good performance of PACS when solving the large scale problems. As the size of job increases, the number of batches formed becomes smaller. For S2 problems (large-size jobs), due to most individual jobs needing to be assigned into one batch, the feasible search space is much smaller compared to the other cases. This led to a reduction in the performance values for PACS. Note that for the problems with large size jobs, although the performance values are not as large as the case of problems with small jobs, the increment in the performance values is more likely as the problem size increases. For example, when solving problems with a large job size, the average $I_D$ value by PACS was decreased to 0.7108, and the average $I_D$ value was increased to 0.2405. As the value of the arrival time factor $\alpha$ increases, the contention for machines gets lighter; as the value of due date factor $\beta$ increases, the allowance time of jobs gets larger. In both of the above conditions, there is more possibility to complete the scheduling of jobs within their due dates, but the performance of the three algorithms is not very different.

\[
\text{GAP} = \frac{I_D^{\text{PACS}} - I_D^{\text{SMOGA}}}{I_D^{\text{PACS}}} \times 100\%
\]

Averaging over all 360 problem instances, the results showed that the average time needed for PACS was longer than SMOGA, especially for large scale problem. Due to the adoption of dynamic heuristic information in PACS, the heuristic information depends on the partial solution constructed so far and therefore has to be computed for each move. This determines a higher computational cost that may be compensated by the higher accuracy of the computed heuristic values for the PACS algorithm. When the number of jobs increases, the size of feasible solution space increases exponentially. Thus the PACS algorithm has a higher computational requirement than the SMOGA algorithm, especially for large scale problem. As can be seen, the average time consumption of SMOGA and PACS were 37 s and 133 s.

Table 7

Performance comparison of the three algorithms on different problem characteristics.

<table>
<thead>
<tr>
<th>Number of jobs(J)</th>
<th>$I_D$</th>
<th>$I_H$</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SMOGA</td>
<td>OPACS</td>
<td>PACS</td>
</tr>
<tr>
<td>20</td>
<td>0.4294</td>
<td>0.2942</td>
<td>0.3218</td>
</tr>
<tr>
<td>50</td>
<td>0.6664</td>
<td>0.3606</td>
<td>0.1150</td>
</tr>
<tr>
<td>100</td>
<td>0.7655</td>
<td>0.3684</td>
<td>0.0998</td>
</tr>
<tr>
<td>200</td>
<td>0.8533</td>
<td>0.3974</td>
<td>0.0948</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Size of job(S)</th>
<th>$I_D$</th>
<th>$I_H$</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.7542</td>
<td>0.2597</td>
<td>0.1116</td>
</tr>
<tr>
<td>2</td>
<td>0.5951</td>
<td>0.3565</td>
<td>0.2407</td>
</tr>
<tr>
<td>3</td>
<td>0.6873</td>
<td>0.4493</td>
<td>0.1213</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Release time of job(R)</th>
<th>$I_D$</th>
<th>$I_H$</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.7263</td>
<td>0.2903</td>
<td>0.1514</td>
</tr>
<tr>
<td>2</td>
<td>0.6630</td>
<td>0.2971</td>
<td>0.1707</td>
</tr>
<tr>
<td>3</td>
<td>0.6475</td>
<td>0.4781</td>
<td>0.1514</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Due date of job(D)</th>
<th>$I_D$</th>
<th>$I_H$</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.6770</td>
<td>0.3458</td>
<td>0.1508</td>
</tr>
<tr>
<td>2</td>
<td>0.6808</td>
<td>0.3645</td>
<td>0.1649</td>
</tr>
</tbody>
</table>

| Average           | 0.6789 | 0.3552 | 0.1578 | 0.1675 | 0.5136 | 0.7948 | 37.40 | 291.63 | 133.74 |

Please cite this article as: Xu, R., et al., A bi-objective scheduling problem on batch machines via a Pareto-based ant colony system. International Journal of Production Economics (2013), http://dx.doi.org/10.1016/j.ijpe.2013.04.053
respective. However, PACS performed much better than SMOGA in both metrics. On the average, the $l_2$ and $l_0$ of our PACS were better than those of SMOGA by 374% and 330% calculated by Eq. (16), respectively. Due to different encoding mechanisms and search techniques, PACS need for greater computational cost to calculate transition probability is outweighed by its capability to explore the entire search space which potentially contains high-quality solutions to obtain a superior performance.

Considering the computational efficiency of the two MOACO algorithms, not only was the overall performance of PACS better than OPACS, but the computation time required by PACS was shorter than as well. Averaging over all 360 problem instances, the $l_2$ and $l_0$ of our PACS were better than those of OPACS by 54.7% and 125% calculated by Eq. (16), respectively, and the average time consumption of PACS and OPACS were 133 s and 291 s, respectively. Clearly, the proposed distinctive features employed in PACS cause a better performance with less computational cost. One source for the speedup is the difference between the two solution construction mechanisms. Another source for the speedup could be the use of candidate list strategy in our algorithm, which helps our PACS algorithm reduce the search space significantly and guide the search toward promising regions in the solution space and results in the faster convergence of PACS.

A major limitation of the proposed PACS algorithm is that, it may not appropriate for solving multi-objective scheduling problem with more than two objectives. This is because PACS constructed dynamic heuristic information based on the problem-specific knowledge of each objective. Although this method allows for a more precise guidance of the search process, it may results in a low computational efficiency when dealing with more objectives.

6. Conclusions

In this paper, we addressed the bi-criteria scheduling problem on a set of BPMs arranged in parallel with non-identical job sizes and dynamic job arrivals for minimizing makespan and maximum tardiness. With the considerations of different constraints and multiple objectives, this problem has a high practical value while it becomes much harder to solve. We proposed a scheduling approach based on the framework of MOACO to solve the target problem. It employed a new solution construction mechanism that considered three decisions (i.e. batch formation, machine assignment, and batch sequencing) simultaneously for encoding a feasible solution that explored the entire solution space. Furthermore, we also developed the suitable components of candidate list and heuristic information corresponding to the new construction mechanism, which could direct the search toward promising regions and provide more precise guidance of the search process. Taking into account different evaluation metrics, computational results showed that the proposed PACS algorithm significantly outperformed the compared algorithms as the number of jobs increased. Future work includes extending our algorithm to the case of BPM scheduling with other constraints and optimization objectives. Also, it is of interest to consider other machine environments, such as flow shop or job shop.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (No.71171184/71201151/61175065), the Funds for Creative Research Group of China (No. 70821001), the NSFC major program (No.71090401/71090400), the Postdoctoral Science Foundation of China (No.2011IM501067), the Fundamental Research Funds for the Central Universities (No. WK0110000032), the Program for New Century Excellent Talents in University (No.NCET-12-0512), the Natural Science Foundation of Anhui Province (No.11080851J16), and the Cai Yuanpei Program of the French Ministries of Foreign and European Affairs and the Chinese Ministry of Education (No.27927VE). This research also received fund from the University of Tennessee Health Information Technology & Simulation (HITS) Laboratory.

Appendix A. Mathematical model

Indexes

- $j$ job index, $j = 1, 2, ..., n$
- $b$ batch index, $b = 1, 2, ..., l$
- $m$ machine index, $m = 1, 2, ..., p$

Parameters

- $B$ capacity of batch machine
- $p_j$ processing time of job $j$
- $r_j$ release time of job $j$
- $d_j$ due date of job $j$
- $s_j$ size of job $j$
- $A$ a large constant number which is at least as large as the maximum completion time of all jobs

Decision variables

- $P_{bm}(i)$ processing time of batch $b$ on machine $m$ of schedule $i$, e.g., $P_{bm}$
- $R_{bm}(i)$ release time of batch $b$ on machine $m$ of schedule $i$, i.e., $R_{bm}$
- $S_{bm}(i)$ starting time of batch $b$ on machine $m$ of schedule $i$, i.e., $S_{bm}$
- $T_{bm}(i)$ tardiness of batch $b$ on machine $m$ of schedule $i$, i.e., $T_{bm}$
- $C_{bm}(i)$ completion time of batch $b$ on machine $m$ of schedule $i$, i.e., $C_{bm}$

Objectives

- $C(\sigma)$ maximum completion time of schedule $\sigma$, i.e., $C_{\max}$
- $T(\sigma)$ maximum tardiness of schedule $\sigma$, i.e., $T_{\max}$

The decision variable $X_{jbm}$ is defined as:

$$X_{jbm} = \begin{cases} 1, & \text{if job } j \text{ is assigned to batch } b \text{ on machine } m \\ 0, & \text{otherwise} \end{cases}$$

Based on the above notations, we develop the following mathematical model:

\begin{align}
\text{Minimize} & \quad \{C_{\max}, T_{\max}\} \\
\text{s.t.} & \quad \sum_{m=1}^{n} \sum_{b=1}^{l} X_{jbm} = 1, \quad j = 1, ..., n \\
& \quad \sum_{j=1}^{n} s_j X_{jbm} \leq B, \quad b = 1, ..., n; m = 1, ..., p \\
& \quad P_{bm} \geq p_j X_{jbm}, \quad j = 1, ..., n; b = 1, ..., n; m = 1, ..., p \\
& \quad S_{bm} \geq r_j X_{jbm}, \quad j = 1, ..., n; b = 1, ..., n; m = 1, ..., p \\
& \quad S_{bm} \geq S_{b-1m} + P_{b-1m}, \quad b = 2, ..., n; m = 1, ..., p \\
& \quad C_{bm} \geq S_{bm} + P_{bm}, \quad b = 1, ..., n; m = 1, ..., p
\end{align}
Note that the number of batches is uncertain before constructing a complete solution. Hence we set the number of batches to be equal to the number of jobs \( n \) in this model. A batch is opened if it contains at least one job; otherwise, it is closed with a processing time of zero. Eq. (1) denotes the objective functions of our presented model: minimizing the makespan and the maximum tardiness. Eq. (2) ensures that each job can only be assigned into exactly one batch on one machine. Eq. (3) guarantees that the total size of all jobs in each batch processed on a machine does not exceed one batch on one machine. Eq. (4) ensures that each job can only be assigned into a complete solution. Hence we set the number of batches to be necessary or unnecessary according to the values of our decision variable \( X_{jbm} \). If job \( j \) has not been assigned to batch \( b \) of machine \( m \), \( X_{jbm} \) yields \( -M \) value in this expression. Because \( M \) is positive with a very big value, the relevant constraint becomes unnecessary in accordance with Eq. (11). Otherwise \( X_{jbm} = 1 \), this third expression yields zero value, and the right side therefore expresses relevant tardiness. Eq. (12) expresses the objective makespan, which equals \( \sum_{m=1}^{M} \max_{b} \left( \sum_{p=1}^{p_{j}} t_{jpbm} - d_{jpm} \right) \). Eq. (13) denotes a binary restriction on the decision variable.

References


