Information Sciences 340–341 (2016) 1–16

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Solving integrated process planning and scheduling problem with constructive meta-heuristics

Luping Zhang [a,b], T.N. Wong [b,*]

[a] Southwestern University of Finance and Economics, Chengdu, China
[b] Department of Industrial and Manufacturing Systems Engineering, The University of Hong Kong, Pokfulam Road, Hong Kong

ARTICLE INFO

Article history:
Received 30 December 2014
Revised 25 September 2015
Accepted 2 January 2016
Available online 8 January 2016

Keywords:
Jobshop scheduling
Process planning
Constructive meta-heuristics
Ant colony optimization

ABSTRACT

For product manufacturing, process planning is to select a series of manufacturing processes according to the product design specification, and scheduling is to allocate manufacturing resources such as machines and tools to these processes. It is a common problem that the process plan and the schedule are not able to cope with the changes in real time manufacturing. Integrated process planning and scheduling (IPPS) is to conduct the process planning and scheduling functions concurrently, with the aim to improve the dynamic responsiveness of the production schedule. This paper investigates the formulation and implementation of constructive meta-heuristics for solving IPPS problems. To begin with, a model representation is established to express IPPS problems with AND/OR graphs. With this model representation, a generic framework is proposed for implementing constructive meta-heuristics in the solution model. The generic framework provides a common procedure for the constructive meta-heuristics, which encapsulates the calculation of the search frontier and state transitions, and provides two interfaces for accommodating different constructive search algorithms. Ant colony optimization (ACO), a commonly-used algorithm which possesses all typical characteristics of constructive meta-heuristics, is adopted as a representative example for illustrating the implementation. Experiments and tests are conducted to validate the proposed system. The single objective minimizing the makespan is set for evaluating the performance of the proposed system. Experimental results of the benchmark problems have shown the effectiveness and high performance of the proposed approach based on the integration of the generic framework and ACO strategy.

1. Introduction

Traditionally, production scheduling and process planning are implemented as two sequential decision-making functions for product manufacturing. The process planning function is responsible for selecting the sequence of manufacturing processes according to the product design specification; scheduling will then allocate manufacturing resources such as machines and tools to the processes, subject to constraints such as release time, due dates, and machine load.

In usual practice, alternate process sequences are generated in the process planning stage. The best sequence is selected as the process plan to serve as input for scheduling. This approach is rigid and it limits the alternatives that may be used in the scheduling stage for global performance improvement. On the other hand, job scheduling based on a fixed process plan may cause severely unbalanced machine loading, leading to superfluous bottlenecks. Worse still, disruptions and changes

* Corresponding author. Tel.: +852 2859 7055; fax: +852 2858 6535.
E-mail address: tnwong@hku.hk (T.N. Wong).

http://dx.doi.org/10.1016/j.ins.2016.01.001
0020-0255/

happen frequently and almost cannot be avoided in practical manufacturing environments. The acceptable schedules yielded at the predictive scheduling stage may become infeasible due to uncertainties [23].

Integrated process planning and scheduling (IPPS) [3] is to bridge the gap between process planning and scheduling to improve the performance of both functions and make the manufacturing system more responsive to dynamic internal and external environments. IPPS problems are mostly NP-hard which are difficult to be solved by traditional analytical methods with bounded computing time. Researchers have therefore turned to approximate methods such as meta-heuristics for solving the IPPS problem. As an alternative solution approach, our research group has investigated on implementing multi-agent system (MAS) in the IPPS field [29,30]. He et al. also reported an agent-based process planning and scheduling system for make-to-order production [11]. More recently, we have achieved to combine MAS and meta-heuristics to solve the IPPS problem and perform dynamic process re-scheduling [40].

Meta-heuristics have been widely used to solve complex combinatorial optimization problems. Based on different classification criteria, meta-heuristics can be classified into different categories. One important classification is: *constructive and improvement meta-heuristics.* Constructive meta-heuristics generate new solutions from an empty or partial solution at each iteration. A typical and widely-used constructive meta-heuristic is the ant colony optimization (ACO). Improvement meta-heuristics start from an initial solution, and iteratively improve the solution quality via neighborhood search. Genetic algorithms (GAs) are a kind of representative improvement meta-heuristics [9].

This paper investigates the approach of applying constructive meta-heuristics for the IPPS problem. A generic framework for the constructive meta-heuristics which is able to accommodate a variety of search strategies is proposed. An ACO is implemented to illustrate the application of constructive meta-heuristic in the generic framework. The rest of the paper is arranged as follows. Implementations of meta-heuristics in IPPS are reviewed in Section 2. Section 3 illustrates a graph model for the IPPS problem solvable by meta-heuristics. Section 4 gives an investigation on the common process of implementing constructive approaches to solve the IPPS problem. Integrating a simple search strategy and ACO into a generic framework as concrete implementations is illustrated in Section 5. Section 6 shows a series of experiments on a benchmark test bed. Conclusions are drawn in Section 7.

2. Literature review

There is a tremendous amount of literature on the respective separate approaches of process planning and scheduling. Regarding process planning, Yildiz made an outstanding contribution to the turning operations optimization problems [34,37] and structural design optimization problems [35,36]. Their contributions also include a structural damage detection approach [7] and a new particle swarm based vehicle crashworthiness optimization approach [38]. In addition, much effort has been devoted to the development of efficient computer-aided process planning (CAPP) systems to link CAD and CAM [2,32,39]. More recently, research on reconfigurable process planning and process plan flexibilities has been intensified to cater for dynamic and flexible manufacturing environment [1,25].

IPPS involves flexibilities of process plans which usually fall into three categories: operation flexibility related to the possibility of performing an operation on different machines with non-identical processing time, sequencing flexibility corresponding to the possibility of reordering the manufacturing operations in the operation sequence, and processing flexibility related to the possibility of adopting different operation sequences to process the same one or group of manufacturing features [14]. Different meta-heuristics have been applied to solve the IPPS problems, for instance, GA [19], simulation-based GA [16], symbiotic evolutionary algorithm (SEA) [14], modified GA [24], particle swarm optimization (PSO) [10], multi-objective GA [43], improved genetic algorithm [21], and two-staged ACO [42]. Besides, Shen and Yao proposed a mathematical modeling and multi-objective evolutionary algorithms for dynamic flexible jobshop scheduling problems [26]. Zhang et al. proposed a hybrid sampling strategy-based multi-objective evolutionary algorithm for the process planning and scheduling problem [44]. Subsequently, a recent article reported on the design and implementation of multi-objective evolutionary algorithms in a variety of manufacturing scheduling problems including IPPS [8].

Ant systems are a computational paradigm proposed for stochastic combinatorial optimization [5] and have been attracting increasing attentions in recent years. Simulating the behavior of ants depositing pheromones along the trail as they search for food around their colony, ant systems have the natural merit to find good paths through graphs. As mediated by the environment, the path is optimized through ants' cooperative interactions [6]. Ant systems for a variety of scheduling problems have been reported in literature. For instance, the first ant system for jobshop scheduling problems (JSPs) [4], ACO approach for project scheduling [18], ACO algorithm for a 2-machine bi-criteria flowshop scheduling problem [27], a Beam-ACO for openshop scheduling [1], an ACO combined with TS for JSPs [13], ACO for multi-objective flowshop scheduling problem [33], and a knowledge-based ACO for flexible jobshop scheduling problem [31]. Regarding job shop scheduling with process planning flexibilities, Rossi reported an ACO approach based on a disjunctive graph model for a jobshop scheduling system with resource flexibility and separable setup times [22]. Kumar et al. [15] reported an ant system for the IPPS problem, but only operation flexibility was considered. Leung et al. [17] covered this gap and proposed an agent-based ACO for the IPPS problems with full consideration of three categories of flexibilities of process plans.

As reviewed in literature, although several commonly-used meta-heuristics and their variants have been adopted as solution approaches for the IPPS problem, a proper IPPS problem model is yet to be defined. Without the formal representation of the IPPS problem, it is difficult to reproduce the solution approaches in similar problem contexts. Moreover, there is still demand for new approaches to jobshop process planning and scheduling problems. Hence in this paper, we attempt to

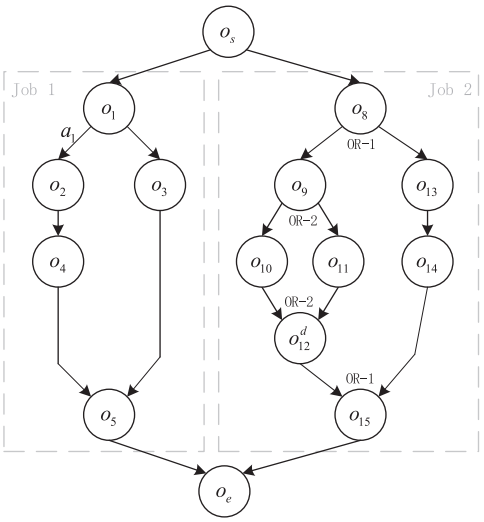

Fig. 1. A sample IPPS instance for illustrative case.

Table 1
Processing data for the sample IPPS instance.

j	1					2						
o_i	o_1	o_2	o_3	o_4	o_5	o_8	o_9	o_{10}	o_{11}	o_{13}	o_{14}	o_{15}
$M(o_i)$	2, 3	1	3, 4	1, 2, 3	2, 4	4	1, 4	1, 3, 4	3	1, 2	3, 4	2, 3
$t(o_i, k)$	10, 15	20	14, 18	7, 8, 6	14, 9	18	9, 11	7, 9, 8	12	13, 17	9, 7	10, 8

establish a generic model for the IPPS problem, and accordingly a general framework for incorporating constructive meta-heuristics in the solution model. The ACO, a commonly-used algorithm which possesses all typical characteristics of constructive meta-heuristics, is adopted as a representative example for illustrating the implementation.

3. The graph model

In this paper, the scope of the IPPS problem is confined to the production of n parts with m machines in the job shop or similar kind of flexible manufacturing environments. It is to select the production operation sequences of all n jobs and allocate the operations to the machines with the objective to optimize the performance criterion such as minimum makespan.

In our previous and ongoing research, AND/OR graphs have been applied to represent IPPS problems and it has been found that meta-heuristics can be easily integrated as the solution strategy [28,41]. A standard AND/OR graph G comprises a set of operations O and a set of directed arcs A. Fig. 1 shows an AND/OR graph representing a sample IPPS instance and Table 1 gives the corresponding processing data. In this paper, operations are labeled with letters $o_1, o_2, \ldots,$ and arcs are labeled with $a_1, a_2, \ldots,$. An arc which connects from one operation to another, is used to represent the precedence relationship between two operations. As shown in Fig. 1, arc a_1 connects from operation o_1 to o_2. The origin operation o_1 is a predecessor of o_2, whilst the tail operation o_2 is a successor of o_1 (denoted $o_1 = \phi(o_2)$). Sometimes, an operation may have multiple predecessors or successors. The two sets of predecessors and successors of operation o_i are denoted $F(o_i)$ and $S(o_i)$ respectively.

The IPPS instance shown in Fig. 1 comprises two jobs with a total of 13 operations to be processed on 4 machines. o_s and o_e are two dummy nodes representing the global start and end points respectively. Job 2 contains a pair of alternative processing trails initiating from o_8 and ending at o_{15}, indicated by "OR-1". One alternative trail of job 2 holds a pair of subordinate alternative process routes indicated by "OR-2". In our graph definition, the subgraph representing an alternative branch is called an or-subgraph. o_8 and o_9, the two operations initiating the or-subgraphs are called or-initiators, whilst o_{12}^d and o_{15}, the two operations ending the or-subgraphs are called or-terminals. All or-branches of the same or-relationship must start from the same or-initiator and end at the same or-terminal. A pair of or-initiator and or-terminal can only enclose or-branches of one or-relationship. More detailed definitions and notations are given in Section 3.1.

Table 2
IPPS elements and notations.

Notation	Description	Notation	Description
j	jth job, $j = 1,\ldots, \boldsymbol{n}$.	$t(o_i, k)$	Processing time of (o_i, k).
k	kth machine, $k = 1, \ldots, \boldsymbol{m}$.	$T^s(o_i, k)$	Start time of (o_i, k).
J	A set of $\boldsymbol{n}$ jobs.	$T^e(o_i, k)$	End time of (o_i, k).
M	A set of $\boldsymbol{m}$ machines.	$T^s(o_i)$	Start time of o_i.
o_i	ith operation.	$T^e(o_i)$	End time of o_i.
(o_i, k)	Processing of o_i on machine k.	$M(o_i)$	The subset of machines capable of processing o_i.
o_i^d	A dummy operation to be processed on a dummy machine ($k = 0$).	O^I	The set of or-initiators.
O_j	The set of operations belonging to job j.	O^T	The set of or-terminals.
$\varphi(o_i)$	An expression to indicate that the job hosts o_i.		

3.1. Elements and notations of the IPPS problem

Table 2 summarizes the elements and notations used in problem modeling and description. Several notations should be highlighted. Firstly, o_i denotes the ith operation, whilst the two-tuple (o_i, k) is to express the processing of o_i on machine k. If a superscript "d" is added to o_i, it indicates a dummy operation which is processed on a dummy machine ($k = 0$) with zero processing time. Secondly, $T^s(o_i)$ and $T^e(o_i)$ represent the start time and end time of o_i. $T^s(o_i)$ and $T^e(o_i)$ are not determined until a specific processing machine is assigned, that is, a (o_i, k) is executed. Finally, in practice, not all machines are capable of processing o_i. In that case, $M(o_i)$ denotes the subset of machines capable of processing o_i.

With the consideration of processing flexibility, a job can be finished by tracing different process trails. However, only one process trail can be selected for execution. It implies that only those operations on selected process trails are needed, whilst the remainder operations of the same job are left unscheduled. A new variable $p(o_i)$ is introduced to indicate whether o_i is selected for execution. Likewise, considering alternative machines, o_i can only be processed on one candidate machine. $p(o_i, k)$, another new variable, is then adopted to express whether o_i is processed on machine k. $p(o_i)$ and $p(o_i, k)$ are subject to the following definitions:

$$p(o_i) = \begin{cases} 1, & \text{if } o_i \text{ is selected and scheduled} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$p(o_i, k) = \begin{cases} 1, & \text{if } (o_i, k) \text{ is executed} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

3.2. Model representation

The complete IPPS solution approach should comprise the selection and sequencing of operations for all n jobs, and then assigning the operations to the m machines for scheduling. In general, meta-heuristics are able to yield a sequence of operations, that is, meta-heuristics can only determine which operation to be selected and in what order to process them. It is then required to assign the selected operations to an empty schedule, the task of calculating time-related variables like start time of each selected operation is dispatched to a function which is called mapping rule in this paper. Hence, with the incorporation of meta-heuristics, the IPPS problem is similar to a sequencing-like problem, that is to select necessary operations for each job and arrange them into a whole sequence.

A variable $\varepsilon(o_{i_1}, o_{i_2})$ is introduced to indicate the precedence relationship of two operations o_{i_1} and o_{i_2} which belong to the same job and are both selected for execution. The definition is as follows:

$$\forall\, o_{i_1}, o_{i_2} \in O, o_{i_1} \neq o_{i_2}, \varphi\left(o_{i_1}\right) = \varphi\left(o_{i_2}\right) = j:$$
$$\varepsilon\left(o_{i_1}, o_{i_2}\right) = \begin{cases} 1, & \text{if } o_{i_1} \text{ precedes } o_{i_2} \\ 0, & \text{if } o_{i_2} \text{ precedes } o_{i_1} \end{cases} \tag{3}$$

Moreover, another variable $\xi(o_i, k)$ is needed to record the locus of (o_i, k) in the selected operation sequence. The IPPS problem can be formulated as an integer programming (IP) model with optimization objectives such as minimum makespan, minimum tardiness, and maximum machine utilization. In general, minimizing the makespan is the most important and commonly-used objective. With regard to our graph model of the IPPS problem, since the global sink operation o_e indicates the end of all jobs, the objective of minimizing the makespan can be easily written as

$$\min\ T^e(o_s)$$

subject to the following constraints:

$$p(o_s) = 1 \tag{4}$$

$$p(o_i) - \sum_{k \in M(o_i)} p(o_i, k) = 0 \quad \forall o_i \in O \tag{5}$$

$$\sum_{o_{i_1} \in S(o_{i_0})} p(o_{i_1}) - p(o_{i_0}) = 0 \quad \forall o_{i_0} \in O^I \tag{6}$$

$$p(o_{i_1}) - p(o_{i_0}) \geq 0 \quad \forall o_{i_0} \in O, o_{i_0} \neq o_e, o_{i_0} \notin O^I, o_{i_0} = \phi(o_{i_1}) \tag{7}$$

$$p(o_{i_0}) - p(o_{i_1}) \geq 0 \quad \forall o_{i_1} \in O, o_{i_1} \neq o_s, o_{i_1} \notin O^T, o_{i_0} = \phi(o_{i_1}) \tag{8}$$

$$\sum_{o_{i_0} \in F(o_{i_1})} p(o_{i_0}) - p(o_{i_1}) = 0 \quad \forall o_{i_1} \in O^T \tag{9}$$

$$\sum_{o_i \in O} p(o_i) - \xi(o_i, k) \geq 0 \quad \forall o_i \in O, k \in M(o_i) \tag{10}$$

$$\xi(o_i, k) - p(o_i, k) \geq 0 \quad \forall o_i \in O, k \in M(o_i) \tag{11}$$

$$W \times p(o_i, k) - \xi(o_i, k) \geq 0 \quad \forall o_i \in O, k \in M(o_i) \tag{12}$$

$$W \times \left(2 - p(o_{i_1}, k_1) - p(o_{i_2}, k_2)\right) + \xi(o_{i_2}, k_2) - \xi(o_{i_1}, k_1) \geq 1$$
$$\forall o_{i_1} \in O, k_1 \in M(o_{i_1}), \forall o_{i_2} \in O, k_2 \in M(o_{i_2}), o_{i_1} = \phi(o_{i_2}) \tag{13}$$

$$W \times \left(3 - p(o_{i_1}, k_1) - p(o_{i_2}, k_2) - \varepsilon(o_{i_1}, o_{i_2})\right) + \xi(o_{i_1}, k_1) - \xi(o_{i_2}, k_2) \geq 1$$
$$\forall o_{i_1} \in O, k_1 \in M(o_{i_1}), \forall o_{i_2} \in O, k_2 \in M(o_{i_2}), o_{i_1} \neq o_{i_2} \tag{14}$$

$$W \times \left(2 - p(o_{i_1}, k_1) - p(o_{i_2}, k_2) + \varepsilon(o_{i_1}, o_{i_2})\right) + \xi(o_{i_2}, k_2) - \xi(o_{i_1}, k_1) \geq 1$$
$$\forall o_{i_1} \in O, k_1 \in M(o_{i_1}), \forall o_{i_2} \in O, k_2 \in M(o_{i_2}), o_{i_1} \neq o_{i_2} \tag{15}$$

$$p(o_i), p(o_i, k), \varepsilon(o_{i_1}, o_{i_2}) \in \{0, 1\} \tag{16}$$

$$\xi(o_i, k) \in N^0 \quad \forall o_i \in O, k \in M(o_i) \tag{17}$$

In constraints (12)–(15), W is a big constant value. Constraint (4) means the release of all jobs. Constraint (5) forces only one machine is assigned to process a selected operation. Constraint (6) takes care of the dependency relationship between an or-initiator and its subsequent or-subgraphs. On one hand, if an or-initiator is selected, one of its successors which is the source operation of an or-subgraph has to be selected. On the other hand, if an or-initiator is not selected, none of its successors can be selected for processing. Constraints (7) and (8) take care of the dependency relationship between two connected operations in case that the predecessor is not an or-initiator or the successor is not an or-terminal. Constraint (9) forces the dependency relationship between an or-terminal and its preceding or-subgraphs. Constraints (10) and (11) restrict the values of $\xi(o_i, k)$ between zero and the length of the sequence. Constraint (12) implies that if a process (o_i, k) is not selected, which will not be included in the sequence, its position index $\xi(o_i, k)$ should equal to zero. Constraint (13) forces that an operation could only be placed after its predecessors. Constraints (14) and (15) regulate that two operations in the sequence cannot share a same position index. Constraints (16) and (17) give the domain of all variables.

4. Incorporating constructive meta-heuristics for the IPPS problem

With the graph model illustrated above, it is then to establish an algorithm to search the AND/OR graph to determine an operation sequence P_g, where g is the index number. We can then calculate the corresponding schedule H_g from P_g with a function:

$$H_g = \mu(P_g) \tag{18}$$

where μ is called the *mapping rule*. (P_g, H_g) or $(P_g, \mu(P_g))$ is a solution to an IPPS problem (denoted by Φ_g). The sequence of operations P_g is written as

$$P_g = \left[(o_{i_1}, k_1), (o_{i_2}, k_2), (o_{i_3}, k_3), \ldots,\right].$$

We use $P_g[i]$ to express the ith operation in P_g, and $len(P_g)$ to return its length (total number of operations in P_g).An easy way to calculate H_g with the mapping rule is to put operations in P_g onto an empty or partial schedule orderly from the beginning to the end to determine values of $T^s(o_i, k), T^e(o_i, k)$ and $T^s(o_i), T^e(o_i)$, with the consideration of precedence and

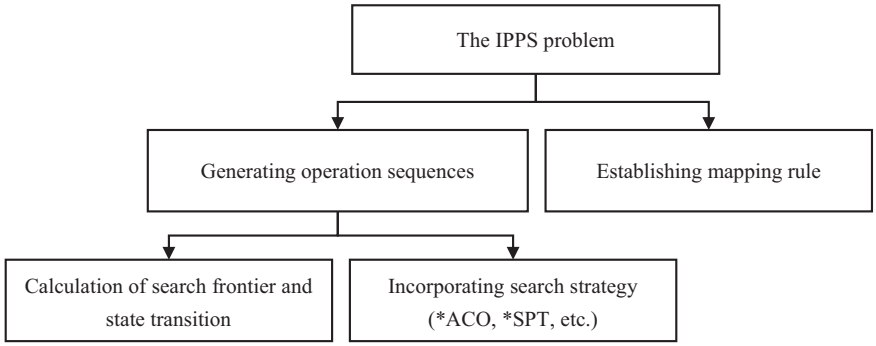

Fig. 2. Decomposition of IPPS system using constructive approaches.
*ACO: ant colony optimization which is a representative constructive metaheuristics. SPT shortest processing time first rule which is a widely-used dispatching rule for scheduling.

no-overlapping constraints. Therefore, an operation in the sequence will definitely be allocated ahead of its succeeding operations. Hence, if an algorithm enables to place an operation after its predecessors in P_g, the feasibility of the corresponding schedule obtained by Eq. (18) can be easily guaranteed.

As discussed above, constructive heuristic approaches are responsible for finding a sequence and determining values for those integer variables given in the IP model in Section 3.2, whilst the task of calculating $T^s(o_i,k), T^e(o_i,k)$ and $T^s(o_i), T^e(o_i)$ is dispatched to the mapping rule. Majority of variables of the IP model are binaries, and domains of the integer variables $\xi_g(o_i,k)$ are restricted to a small interval. Hence, the proposed IPPS model is a relatively simple model which can be easily settled by search-based algorithms.

4.1. A generic framework for constructive approaches

Constructive approaches are supposed to start from an empty or partial sequence and put in operations iteratively while the algorithm operator searches on the AND/OR graph. There are two main issues in the graph-based search: one is how to distinguish those operations the algorithm can visit in each subsequent step, and the other is how to make a decision to move from one operation to another.

In this paper, an operation which could be visited at a step during the search is called a *reachable* operation. All reachable operations form a set named *search frontier*. The approach to determine a movement from one operation to another is called the *search strategy*. Accordingly, the two issues can be briefed as calculating the search frontier and establishing the search strategy.

Adopting constructive approaches as the solution strategy, the IPPS problem can be decomposed into three sub-problems as shown in Fig. 2. A generic framework with well-defined functions for search frontier calculation and state transition could be established. Such framework is able to accommodate a variety of search strategies and mapping rules to facilitate the implementation of different constructive approaches.

In following discussions, the search frontier and search strategy are denoted by O^F and Γ, respectively.

4.1.1. Initial status and main procedure

Initially, the system stays at a status where: the operation sequence is empty, that is $P_g = []$; $\forall o_i \in O : p(o_i) = 0$; only the global source operation is reachable initially, that is $O^F = \{o_s\}$; none of the operations is selected and each operation has not been assigned with processing machine, that is $\forall o_i \in O, k \in M(o_i) : p(o_i,k) = 0$.

The following main steps are carried out to search a complete sequence of operations for all jobs.

(1) Apply a search strategy Γ to select an operation o_i from O^F, and assign a processing machine k as well.
(2) Put (o_i,k) to the end of P_g, and let $p(o_i,k) = 1,\ p(o_i) = 1$.
(3) Check whether the global sink operation is reached. If $o_i = o_e$, terminate the search and return the sequence P_g. Or recalculate O^F and go into the next iteration.

It is not necessary for individual constructive methods to calculate the search frontier and the relevant state transition, which is to be handled in the main procedure.

(a) Calculating the search frontier

With regard to the reachability, there are three kinds of operations which should be investigated separately: or-terminals, the source operation of or-subgraphs, and operations which belong to neither of above two kinds.

Based on the definition of conjunction, disjunction and precedence relationship the reachability of an operation should be discussed in following aspects: if o_i is an or-terminal, it can be visited after one of its predecessors has been selected;

if o_i is neither an or-terminal nor the source operation of an or-subgraph, it can be visited only after all of its predecessors has been selected; if o_i is the source operation of an or-subgraph, it can be visited when none of the source operations on correlated or-subgraphs has been visited. Accordingly, we can write

$$\forall o_i \in O,\ p(o_i) = 0 : o_i \text{ is reachable if} \begin{cases} \sum_{o_{i_0} \in F(o_i)} p\left(o_{i_0}\right) = 1 \ if\ o_i \in O^T \\ \prod_{o_{i_0} \in F(o_i)} p\left(o_{i_0}\right) = 1 \ if\ o_i \notin O^T \wedge \forall \phi(o_i) \notin O^I \\ \sum_{o_{i_0} \in S(\phi(o_i))} p\left(o_{i_0}\right) = 0 \wedge p(\phi(o_i)) = 1 \ if\ \forall \phi(o_i) \in O^I \end{cases} \tag{19}$$

(b) The state transition

Every time an operation is selected, the reachability of a small set of operations is affected. Recalculating the reachability of all operations at each step is unnecessary if a well-defined state transition approach is incorporated. The state transition is to remove unreachable operations out of O^F, and recalculate the reachability of successors of the selected operation as well as putting reachable successors into O^F.

The generic framework also provides two interfaces to integrate different search strategies and mapping rules for various constructive approaches.

4.1.2. The generic framework

System flow of the generic framework is outlined in the following:

Generic framework

Step 1. Initialization. Assign an index number g to the empty sequence.
Step 2. Select an operation from O^F and assign a processing machine.
  Step 2.1. Construct a selection pool $\{(o_i, k) | o_i \in O^F, k \in M(o_i)\}$, and apply Γ to select one item from it. Say (o_i, k) is selected.
  Step 2.2. Update the status of affected operations, that is, let $p(o_i) = 1$, $p(o_i, k) = 1$. Put (o_i, k) to the end of P_g.
  Step 2.3. If the global sink operation is reached, or say $o_i = o_e$, terminate! Else go to the next step.
Step 3. The state transition.
  Step 3.1. Remove the selected operation out of the search frontier, that is $O^F \leftarrow O^F \backslash \{o_i\}$.
  Step 3.2. If the selected operation is the source operation of an or-subgraph, all source operations of all correlated or-subgraphs should be removed. That is, if $|F(o_i)| = 1 \wedge \phi(o_i) \in O^I$,$O^F \leftarrow O^F \backslash \{o_{i_0} | o_{i_0} \in O^F, \phi(o_{i_0}) = \phi(o_i)\}$.
  Step 3.3. Add new reachable operations into the search frontier. $O^F \leftarrow O^F \cup \{o_{i_0} | o_{i_0} \in S(o_i), o_{i_0}$ is reachable according to (19)$\}$.
Step 4. Go to Step 2.

In Step 2.1, a search pool is constructed to accommodate operations of processing every reachable operation on all its candidate machines, say $\{(o_i, k) | o_i \in O^F, k \in M(o_i)\}$. It is to simultaneously fulfill the selection of an operation from the search frontier and assignment of a processing machine.

4.2. An example

Here we use an IPPS instance shown in Fig. 1 and Table 1 as an example to illustrate how to incorporate solid mapping rules and search strategies into the generic framework and investigate how the constructive algorithm works.

In the example, the *shortest processing time first (SPT)* rule is adopted as the search strategy. A simple mapping rule is also established. To illustrate, the following notations are used:

C_k^M: the current completion time of machine k;
C_j^J: the current completion time of job j;
$C^{\max}$: the makespan.

Mapping rule $\mu(P_g)$

(1) Initialization: let the schedule H_g be empty; $\forall k \in M : C_k^M = 0$; $\forall j \in J : C_j^J = 0$.
(2) Map (o_i, k):
  (a) Calculate the start time and end time of the operation by

$$\begin{aligned} T^s(o_i) &= T^s(o_i, k) = \max\left\{C_k^M, C_{\varphi(o_i)}^J\right\} \\ T^e(o_i) &= T^e(o_i, k) = T^s(o_i, k) + t(o_i, k) \end{aligned} \tag{20}$$

Table 3
Summary of the search procedure on the illustrative case.

Round	O^F	Selection pool and selection criterion $(o_i,k)\|_{t(o_i,k)}$	Selected (o_i,k)	Deleted o_i	Added o_i
1	o_s	$(o_s,0)\|_0$	$(o_s,0)$	o_s	o_1, o_8
2	o_1, o_8	$(o_1,2)\|_{10}$, $(o_1,3)\|_{15}$, $(o_8,4)\|_{18}$	$(o_1,2)$	o_1	o_2, o_3
3	o_8, o_2, o_3	$(o_8,4)\|_{18}$, $(o_2,1)\|_{20}$, $(o_3,3)\|_{14}$, $(o_3,4)\|_{18}$	$(o_3,3)$	o_3	-
4	o_8, o_2	$(o_8,4)\|_{18}$, $(o_2,1)\|_{20}$	$(o_8,4)$	o_8	o_9, o_{13}
5	o_2, o_9, o_{13}	$(o_2,1)\|_{20}$, $(o_9,1)\|_9$, $(o_9,4)\|_{11}$, $(o_{13},1)\|_{13}$, $(o_{13},2)\|_{17}$	$(o_9,1)$	o_9, o_{13}	o_{10}, o_{11}
6	o_2, o_{10}, o_{11}	$(o_2,1)\|_{20}$, $(o_{10},1)\|_7$, $(o_{10},3)\|_9$, $(o_{10},4)\|_8$, $(o_{11},3)\|_{12}$	$(o_{10},1)$	o_{10}, o_{11}	o_{12}^d
7	o_2, o_{12}^d	$(o_2,1)\|_{20}$, $(o_{12}^d,0)\|_0$	$(o_{12}^d,0)$	o_{12}^d	o_{15}
8	o_2, o_{15}	$(o_2,1)\|_{20}$, $(o_{15},2)\|_{10}$, $(o_{15},3)\|_8$	$(o_{15},3)$	o_{15}	-
9	o_2	$(o_2,1)\|_{20}$	$(o_2,1)$	o_2	o_4
10	o_4	$(o_4,1)\|_7$, $(o_4,2)\|_8$, $(o_4,3)\|_6$	$(o_4,3)$	o_4	o_5
11	o_5	$(o_5,2)\|_{14}$, $(o_5,4)\|_9$	$(o_5,4)$	o_5	o_e
12	o_e	$(o_e,0)\|_0$	$(o_e,0)$	end	-

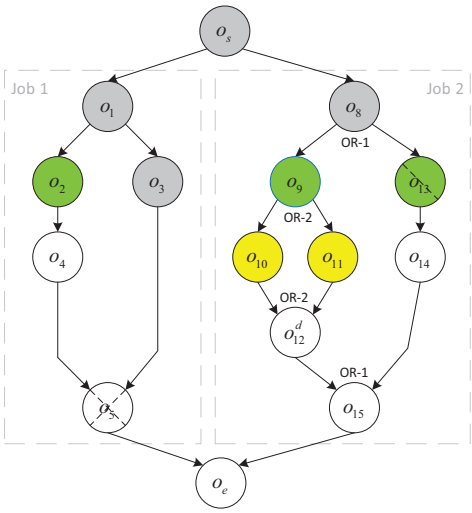

Fig. 3. Illustration of Round 5 of the example incorporating the SPT rule.

(b) Update C_k^M and C_j^J by

$$C_k^M = C_{\varphi(o_i)}^J = T^e(o_i) \tag{21}$$

Table 3 summarizes the search procedure of the algorithm, and the step by step work of the algorithm is described as follows.

Fig. 3 shows a representative round No. 5 during the search procedure of the illustrative case. Gray nodes (o_s,o_1,o_3,o_8) are those operations selected in previous rounds, whilst green nodes (o_2, o_9, o_{13}) are currently reachable operations according to (19). At the beginning of round 5, o_5 is not reachable since o_5 is neither an or-terminal nor a source operation of an or-subgraph, while $\prod_{o_i \in F(o_5)} p(o_i) = p(o_4) \times p(o_3) = 0 \times 1 = 0$.

The selection pool constructed on the search frontier is $\{(o_2,1),(o_9,1),(o_9,4),(o_{13},1),(o_{13},2)\}$. According to the SPT rule, $(o_9,1)$ which has the shortest processing time is selected. Let $p(o_9) = p(o_9,1) = 1$. It is found that the cardinality of $F(o_9)$ $|F(o_9)| = 1$ and $\phi(o_9) = o_8 \in O^J$, implying that o_9 is the source operation of an or-subgraph. Hence according to the state transition rule, $O^F = \{o_2, o_9, o_{13}\} \backslash \{o_i | o_i \in O^F, \phi(o_i) = o_8\} = \{o_2\}$, which implies that o_9 and the source operation of the counterpart or-subgraph o_{13} are removed out of the search frontier. Meanwhile, considering successors of o_9, o_{10} and o_{11}, we have $\prod_{o_i \in F(o_{10})} p(o_i) = \prod_{o_i \in F(o_{11})} p(o_i) = p(o_9) = 1$. Hence, both o_{10} and o_{11} are reachable and needed to be added to the search frontier. That is, $O^F = \{o_2\} \cup \{o_{10}, o_{11}\} = \{o_2, o_{10}, o_{11}\}$.

Explanation of other rounds are not elaborated here. Finally we can get a complete sequence

$$P_g = [(o_s,0),(o_1,2),(o_3,3),(o_8,4),(o_9,1),(o_{10},1),\left(o_{12}^d,0\right),(o_{15},3),(o_2,1),(o_4,3),(o_5,4),(o_e,0)]$$

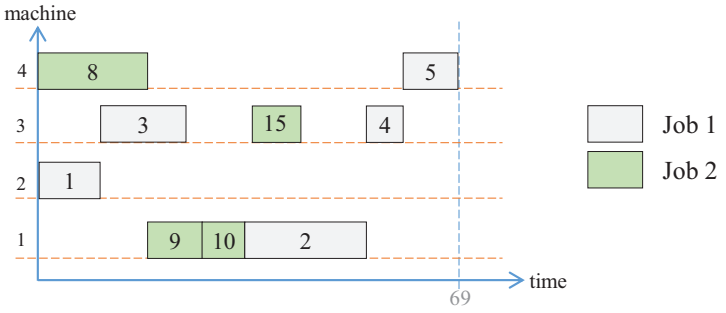

Fig. 4. The final schedule for the illustrative case.

which contains adequate operations to fulfill both jobs of the IPPS instance. The schedule obtained with the simple mapping rule is shown in Fig. 4. The number on each schedule block represents the index of an operation.

5. Incorporating ACO for the IPPS

Integrating ACO into the generic framework to cope with IPPS is discussed in this section. To achieve the integration, exact master strategy, search strategy and mapping rule should be well defined.

5.1. The master strategy

In ant systems, communication between ants is mediated by pheromones deposited on the trail when ants find food and return to the colony. The pheromone quantity on the trail which is frequently traversed will be increased. However, pheromones on every trial will evaporate over time. With the subsequent deposition and evaporation, the pheromone density becomes higher on the more frequently visited paths. It is more likely for ants to follow the trail with higher pheromone density. This is the master strategy of ACO.

Integrating ACO to the generic framework, the ant can traverse through some dummy arcs to find a trail which corresponds to a sequence [17]. Dummy arcs are added to connect two operations (o_{i_1}, k_1) to (o_{i_2}, k_2) (denoted by $< (o_{i_1}, k_1), (o_{i_2}, k_2) >$) where

$$\forall o_{i_1} \in O, \forall k_1 \in M(o_{i_1}), \forall o_{i_2} \in O, \forall k_2 \in M(o_{i_2}), o_{i_1} \neq o_{i_2}, o_{i_2} \neq \phi(o_{i_1}).$$

Dotted arcs in Fig. 5 show the trail of an ant. Each dummy arc actually connects (o_i, k) other than o_i of the graph. For clearer viewing, such kind of dummy arcs and (o_i, k) are not displayed in the AND/OR graph.

To address the IPPS problem with ACO, a colony of ants search on the graph iteratively. Once an ant, say No. g, finds a complete sequence P_g, dummy arcs traversed by ant No. g can be written as

$$\{< P_g[i],\ P_g[i+1] > | i = 1,\ 2, \ldots, len(P_g) - 1\}.$$

The ant deposits a given amount of pheromones $\Delta\tau_g((o_{i_1}, k_1), (o_{i_2}, k_2))$ on every dummy arc on its trail. Pheromones updated by this ant is calculated by

$$\Delta\tau_g\big((o_{i_1}, k_1), (o_{i_2}, k_2)\big) \tag{22}$$

$$= \begin{cases} \dfrac{Q}{L_g - B}, & if\big(o_{i_1}, k_1\big) \in \{P_g[i] | i = 1,\ 2, \ldots, len(P_g) - 1\}, \big(o_{i_2}, k_2\big) = P_g[\xi_g\big(o_{i_1}, k_1\big) + 1] \\ 0, & \text{otherwise} \end{cases} \tag{22}$$

where L_g is the makespan of the solution generated by the ant No. g. Q and B are two positive parameters to control the pheromone amount laid on the trail at each time.

At each iteration, each ant deposits pheromones separately on its trail. At the same time, pheromones will evaporate automatically, hence the total amount of pheromone on dummy arcs is updated by

$$\tau\big((o_{i_1}, k_1), (o_{i_2}, k_2)\big) \leftarrow (1 - \rho) \times \tau\big((o_{i_1}, k_1), (o_{i_2}, k_2)\big) + \sum_{g \in AC} \Delta\tau_g\big((o_{i_1}, k_1), (o_{i_2}, k_2)\big) \tag{23}$$

where ρ is the pheromone evaporation rate and AC is the ant colony.

The pheromone density on each arc will guide the ants searching towards a common direction and hopefully converging to a good trail.

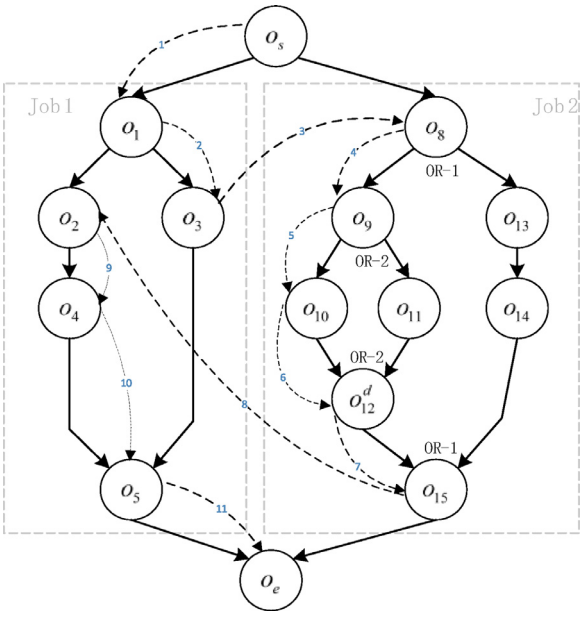

Fig. 5. Trail of the ant.

5.2. Search strategy

In ant systems, each ant applies a probabilistic choice search strategy, also known as *random proportional* rule, to decide the next movement from one operation to another [6]. Particularly, the probability $\vartheta_g((o_{i_1}, k_1), (o_{i_2}, k_2))$ with which the ant g, currently at operation (o_{i_1}, k_1), goes to operation (o_{i_2}, k_2) is

$$\vartheta_g\left(\left(o_{i_1}, k_1\right), \left(o_{i_2}, k_2\right)\right) = \begin{cases} \dfrac{\left[\tau\left(\left(o_{i_1}, k_1\right), \left(o_{i_2}, k_2\right)\right)\right]^{\alpha}\left[\eta\left(\left(o_{i_1}, k_1\right), \left(o_{i_2}, k_2\right)\right)\right]^{\beta}}{\sum_{o_i \in O^F, k \in M(o_i)}\left[\tau\left(\left(o_{i_1}, k_1\right), (o_i, k)\right)\right]^{\alpha}\left[\eta\left(\left(o_{i_1}, k_1\right), (o_i, k)\right)\right]^{\beta}}, & if\ o_{i_2} \in O^F, k_2 \in M\left(o_{i_2}\right) \\ 0, & \text{otherwise} \end{cases} \tag{24}$$

where $\eta((o_{i_1}, k_1), (o_i, k))$ is the heuristic desirability from (o_{i_1}, k_1) to (o_i, k), α and β are the two coefficients used to determine the relative importance of pheromones and heuristic desirability. Various types of visibility can be incorporated for calculating ants' heuristic desirability, such as SPT, length of unscheduled tasks left on machine (TLM), etc. [12]. This paper cooperates the *earliest finishing time* (EFT) as the ant visibility. The finishing time of an operation can be easily obtained if an operation is mapped onto the schedule immediately after it is selected. However, the EFT cannot be used directly since the finishing time of each operation ranges from 0 to the makespan. It is necessary to transform the finishing time of each operation into a bounded range $[B^L, B^U]$ where B^L and B^U are the lower bound and upper bound respectively.

Let $SP(O^F) = \{(o_i, k) | o_i \in O^F, k \in M(o_i)\}$ be the selection pool constructed on O^F, and $T^{\min}$ and $T^{\max}$ be minimum and maximum finishing time of every operation in $SP(O^F)$ if they are tentatively mapped on to the schedule, that is

$$\begin{cases} T^{\min} = \min\limits_{(o_i,k) \in SP\left(O^F\right)} \left(T^e(o_i, k)\right) \\ T^{\max} = \max\limits_{(o_i,k) \in SP\left(O^F\right)} \left(T^e(o_i, k)\right) \end{cases} \tag{25}$$

The bounded finishing time values (or called *fitness value*) for all operations in the selection pool can be calculated by

$$\forall (o_i, k) \in SP\left(O^F\right) : f(o_i, k) = \begin{cases} B^L + \dfrac{B^U - B^L}{T^{\max} - T^{\min}} \times \left(T^e(o_i, k) - T^{\min}\right), & if\ T^{\max} \neq T^{\min} \\ \dfrac{\left(B^L + B^U\right)}{2}, & if\ T^{\max} = T^{\min} \end{cases} \tag{26}$$

And $\eta((o_{i_1}, k_1), (o_i, k))$ in this paper can be written as

$$\eta\left(\left(o_{i_1}, k_1\right), (o_i, k)\right) = \frac{C}{f(o_i, k)} \tag{27}$$

where C is a positive coefficient used to control impact significance of the heuristic desirability.

Table 4
Elements and notations for the enhanced mapping rule.

Notation	Description	Notation	Description
$w^M_{k,i}$	A time window of machine k with index number i.	w_i	A joint time window.
TW^M_k	The set of all time windows of machine k.	$B^L(w)$	Lower bound of the time window w.
$w^J_{j,i}$	A time window of job j with index number i.	$B^U(w)$	Upper bound of the time window w.
TW^J_j	The set of all time windows of job j.	$TW(o_i,k)$	The set of joint time windows which are capable of holding schedule block for (o_i,k).

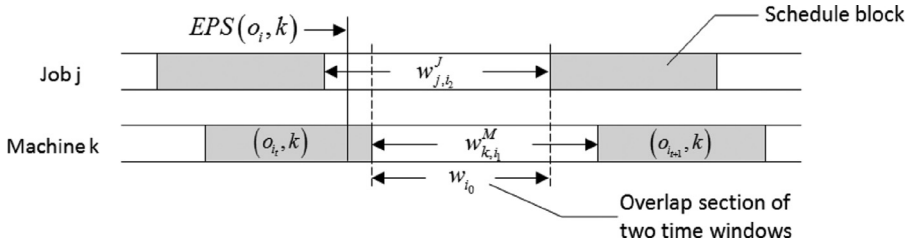

Fig. 6. Overlap section of two time windows.

5.3. An enhanced mapping rule

The example in Section 4.2 incorporates a simple mapping rule, which can be easily implemented but is not effective, especially in terms of minimizing the makespan. It is most likely to leave some vacancies if an unscheduled operation is simply attached to the end of the schedule. However, sometimes it is possible to insert unscheduled operations into these "vacancies" without violating precedence and no-overlapping constraints. The approach of inserting unscheduled operations into "vacancies" is called *time-window based mapping* in this paper.

Table 4 summarizes the elements and notations for the enhanced mapping rule. There are three kinds of time windows. Firstly, If there is a vacancy between two adjacent schedule blocks on machine k, it is called a time window of machine k (denoted $w^M_{k,i}$). Secondly, schedule blocks for operations of job j form a sequence. The vacancy between two adjacent schedule blocks of job j is called a time window of job j (denoted $w^J_{j,i}$). Finally, a new schedule for operation (o_i,k) can only be placed in an overlapping section of w^M_{k,i_1} and w^J_{j,i_2}, in order not to violate the no-overlapping constraints for both jobs and machines. Hence, the overlapping section of a time window for a job and a time window for a machine is called a joint time window (denoted $w_{i_0}=w^M_{k,i_1}\cap w^J_{j,i_2}$). A new schedule block is placed in such a joint time window. Examples of the three kinds of time windows can be found in Fig. 6. In Table 4, the variable w in $B^L(w)$ and $B^U(w)$ is an abstract expression of a time window, which can be $w^M_{k,i}$, $w^J_{j,i}$, or w_i in the calculation.

The following sections describe the functioning of the enhanced mapping rule.

5.3.1. Get available time windows for allocating new schedule blocks

In order to fulfill no-overlapping constraints for both jobs and machines, an unscheduled operation (o_i,k) can only be inserted into a joint time window. As shown in Fig. 6, w_{i_0} is the joint time window of w^J_{j,i_2} and w^M_{k,i_1}. The term $EPS(o_i,k)$ is the *earliest possible starting* time of (o_i,k) determined by precedence constraints. Obviously there are three possible relationships between $EPS(o_i,k)$ and w_{i_0}, which are $EPS(o_i,k)<B^L(w_{i_0})$ when w_{i_0} is on the left side of $EPS(o_i,k)$, $B^L(w_{i_0})\le EPS(o_i,k)\le B^U(w_{i_0})$ when w_{i_0} crosses $EPS(o_i,k)$, and $EPS(o_i,k)>B^U(w_{i_0})$ when w_{i_0} is on the right side of $EPS(o_i,k)$. But no matter what relationship they occupy, the schedule block for (o_i,k) can be placed into w_{i_0} if time interval $B^U(w_{i_0})-\max(EPS(o_i,k),B^L(w_{i_0}))$ is longer than $t(o_i,k)$.

The lower and upper bounds of the joint time window $w_{i_0}=w^M_{k,i_1}\cap w^J_{j,i_2}$ can be calculated by

$$\begin{cases} B^L(w_{i_0}) = \max\left(B^L(w^M_{k,i_1}), B^L(w^J_{j,i_2})\right) \\ B^U(w_{i_0}) = \min\left(B^U(w^M_{k,i_1}), B^U(w^J_{j,i_2})\right) \end{cases} \tag{28}$$

And then, it is easy to obtain the set of available time windows for (o_i,k) with

$$TW(o_i,k)=\left\{ w_{i_0}=w^M_{k,i_1}\cap w^J_{j,i_2} \,|\, \forall w^M_{k,i_1}\in TW^M_k, \forall w^J_{j,i_2}\in TW^J_j, \; B^U(w_{i_0})-\max\left(EPS(o_i,k),B^L(w_{i_0})\right)\ge t(o_i,k)\right\} \tag{29}$$

5.3.2. Allocate the new schedule block

If it is required to process (o_i, k) as early as possible, its schedule block should be allocated in the available time window with the smallest lower bound, that is w_{i_t} such that

$$B^L\left(w_{i_t}\right) = \min_{w_{i_0} \in TW(o_i,k)} \left(B^L\left(w_{i_0}\right)\right) \tag{30}$$

and let

$$T^s(o_i, k) = \max\left(EPS(o_i, k), B^L\left(w_{i_0}\right)\right) \tag{31}$$

5.3.3. Update time windows

After allocating the schedule block for (o_i, k) in the joint time window w_{i_t} and assume $w_{i_t} = w^M_{k,i_1} \cap w^J_{j,i_2}$, it is most likely that new time windows for job j or machine k will be created.

Specifically, if $B^L(w^M_{k,i_1}) < \max(EPS(o_i, k), B^L(w_{i_t}))$, a new time window for machine k, say w^M_{k,i_1+1}, will emerge. That is

$$\begin{cases} B^L\left(w^M_{k,i_1+1}\right) = B^L\left(w^M_{k,i_1}\right) \\ B^U\left(w^M_{k,i_1+1}\right) = \max\left(EPS(o_i, k), B^L\left(w_{i_t}\right)\right) \end{cases} \tag{32}$$

If $w^J_{j,i_2} < \max(EPS(o_i, k), B^L(w_{i_t}))$, a new time window for job j, say w^J_{j,i_2+1}, will emerge. That is

$$\begin{cases} B^L\left(w^J_{j,i_2+1}\right) = B^L\left(w^J_{j,i_2}\right) \\ B^U\left(w^J_{j,i_2+1}\right) = \max\left(EPS(o_i, k), B^L\left(w_{i_t}\right)\right) \end{cases} \tag{33}$$

Then, both w^M_{k,i_1} and w^J_{j,i_2} should be updated with

$$B^L\left(w^M_{k,i_1}\right) = B^L\left(w^J_{j,i_2}\right) \leftarrow B^L\left(w_{i_t}\right) \tag{34}$$

After doing so, if $B^L(w^M_{k,i_1}) = B^U(w^M_{k,i_1})$, w^M_{k,i_1} no longer exists and should be eliminated out of TW^M_k, and so should be w^J_{j,i_2}.

5.3.4. Mapping P_g with the enhanced mapping rule

Step 1. Initialization.

Create an initial time window for each machine: $\forall k \in M : w^M_{k,0}, B^L(w^M_{k,0}) = 0, B^U(w^M_{k,0}) = W$.
Create an initial time window for each job: $\forall j \in J : w^J_{j,0}, B^L(w^J_{j,0}) = 0, B^U(w^J_{j,0}) = W$.
Create a variable $x = 1$.

Step 2. Map an operation onto the schedule.

Get the first unmapped operation in the schedule: $(o_i, k) = P_g[x], j = \varphi(o_i)$.
Calculate joint time windows available for the operation $TW(o_i, k)$ using (28) and (29).
Allocate the schedule block for (o_i, k) according to (30) and (31).

Step 3. Update time windows.

Create time window for machine k using (32), and add it to TW^M_k.
Create time window for job j using (33) and add it to TW^J_j.
Update infected time windows using (34).

$$x \leftarrow x + 1.$$

Step 4. If $x > len(P_g)$, terminate. Else go to Step 2.

5.4. Experiment on the sample IPPS instance

The sample IPPS instance proposed in Section 3 is settled by the system incorporating ACO and the enhanced mapping rule discussed in Section 5.3. Minimizing the makespan is deemed as the optimization objective. The parameter setting for ACO varies case by case and depends on the practical problems. For instance, the number of ants is closely related to the problem scale, too few ants cannot converge to a good path or too many ants will quickly converge to an inferior path. The convergence rate is also related to the amount of pheromone deposited each time and the evaporation ratio. In the experiment, the following parameter combination was found by trial and error to cope with the sample IPPS instance: number of ants = 100; $\alpha = 2$; $\beta = 4$; $C = 100$; $Q = 100$, $B = 300$; $\rho = 0.09$; $B^L = 10$, $B^U = 35$; the original, maximum and minimum amount of pheromones on each dummy arc were set to 1, 20 and 0.1, respectively. The setting of termination

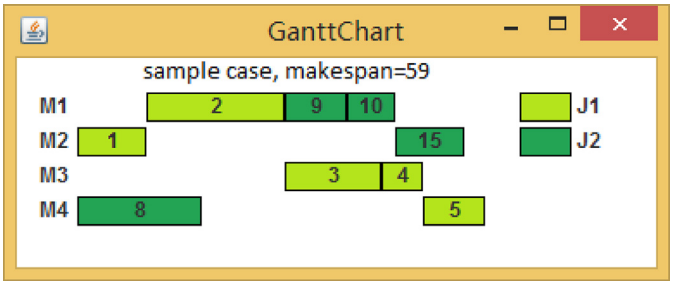

Fig. 7. The schedule generated by ACO for the sample IPPS instance. Numbers on schedule blocks are the index number of corresponding operations.

criteria is to balance the algorithm performance and the computing resources. In our conducted tests, the proposed approach was able to converge to near-optimal solutions within 30 iterations. Hence a simple termination criterion was installed, that is, the maximum number of iterations was set to 30.

The ant system finds a solution with makespan = 59 in 1.8 s, and the corresponding schedule is presented in Fig. 7.

6. Benchmark tests and discussions

For benchmark purpose, the large-scale IPPS problem proposed by Kim et al. is adopted for the experiments [14]. The test bed is composed of 18 separate jobs including 300 operations. The 18 jobs are combined in different ways to form 24 problems which Figures as different kinds and strength of flexibilities.

The proposed ACO is tested on a computer equipped with Intel® Core I-5 CPU (M 520 works@2.4 GHz) and 6GB of RAM. The system is programmed with JAVA, and the adopted database management system is MYSQL Community server (version 5.6.19). Minimizing the makespan is deemed as the objective for optimization.

The benchmark problems are settled by the system incorporating ACO and the enhanced mapping rule discussed in Section 5. The parameter setting is as follows: number of ants = 100; $\alpha = 2$; $\beta = 4$; $C = 100$; $Q = 100$, $B = 300$; $\rho = 0.09$; $B^L = 10$, $B^U = 35$; the original, maximum and minimum amount of pheromones on each dummy arc equal to 1, 20 and 0.1 respectively.

As the termination criteria in our experiments, the number of iterations will not exceed a specific total number of runs. After a series of trial and error, we find that 30 iterations are enough for the ACO to get satisfactory results. Experiments have been conducted with higher number of iterations but results show that the ACO system is hard to find better solutions after 30 iterations. It is considered not worth to spend extra time and computing resources for only a slight improvement, if any, in the solution quality. Hence in our experiments, the proposed ACO runs for a total number of 30 iterations and the best result with shortest makespan among 30 runs is recorded.

A series of experiments are conducted on all 24 problems, experiments on the same problem are repeated 11 times. Best and mean results are recorded and compared with other 3 algorithms: the cooperative co-evolutionary genetic algorithm (CCGA) [20], symbiotic evolutionary algorithm (SEA) [14] and an improved genetic algorithm (IGA) [21]. All results for compared algorithms are taken directly from their respective references.

Table 5 lists the best makespans found for each problem by these 4 algorithms. It shows that the ACO reaches low bounds for 9 problems. Since the optimal value cannot be smaller than the low bound, it also means that the ACO find optimal solutions for those 9 problems. Besides, the ACO yields the best results for other 7 problems. Hence in conclusion, Table 5 reveals the wonderful heuristic ability of the ACO in the IPPS problem domain.

Table 6 accounts for the mean makespans for every problem. The improved rate is the relative improved ratio of the ACO results compared to the minimum results yielded by the other three algorithms. A positive improved rate indicates that the ACO gets best results for corresponding problem. It shows that the ACO outperforms other three algorithms in solving 18 out of 24 problems. Moreover, the ACO performs even better in solving more complex problems such as problem No. 19 to No. 24. What should be highlighted is that the ACO is capable to yield very good solutions for problem No. 24, the most complex problem of the test bed, within 40 s and requires only 30 iterations. Standard Deviations of each problems in ACO experiments reveal a great statistical significance of the ACO.

Compared with CCGA, SEA and IGA, the AND/OR graph is used to model the IPPS problem in this paper. Different from enumerating separated process plans as the input, the AND/OR graph based representation is more flexible for constructive algorithms to find a path between the global source and sink operations. To intensively search an area, it may just need to change the trail between any two points on its original path. The AND/OR graph establishes a global search space, which should be the main reason why the ACO performs better.

Encapsulating the calculation of search frontier and state transition in the generic framework guarantees correctness of the every movement of accommodated algorithms. Or say, as long as the algorithms follow the overall procedure defined by the framework and incorporates a valid mapping rule, the search is definitely restricted in the feasible solution space, and no validation and adjustment rules are needed to guarantee the correctness of the outputs. It may be the main reason for the high efficiency of ACO in our experiments.

Table 5
Comparison of best makespans in benchmark test (ACO).

Problem	Jobs	CCGA	SEA	IGA	ACO	Low bound
1	1, 2, 3, 10, 11, 12	458	428	427	427	427
2	4, 5, 6, 13, 14, 15	363	343	343	343	343
3	7, 8, 9, 16, 17, 18	366	347	344	344	344
4	1, 4, 7, 10, 13, 16	312	306	306	307	306
5	2, 5, 8, 11, 14, 17	327	319	304	318	304
6	3, 6, 9, 12, 15, 18	476	438	427	427	427
7	1, 4, 8, 12, 15, 17	378	372	372	372	372
8	2, 6, 7, 10, 14, 18	363	343	342	343	342
9	3, 5, 9, 11, 13, 16	464	428	427	427	427
10	1, 2, 3, 5, 6, 10, 11, 12, 15	476	443	427	427	427
11	4, 7, 8, 9, 13, 14, 16, 17, 18	410	369	368	364	344
12	1, 4, 5, 7, 8, 10, 13, 14, 16	360	328	312	332	306
13	2, 3, 6, 9, 11, 12, 15, 17, 18	498	452	429	427	427
14	1, 2, 4, 7, 8, 12, 15, 17, 18	420	381	386	382	372
15	3, 5, 6, 9, 10, 11, 13, 14, 16	482	434	427	427	427
16	1, 2, 3, 4, 5, 6, 10, 11, 12, 13, 14, 15	512	454	433	438	427
17	4, 5, 6, 7, 8, 9, 13, 14, 15, 16, 17, 18	466	431	415	398	344
18	1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17	396	379	364	378	306
19	2, 3, 5, 6, 8, 9, 11, 12, 14, 15, 17, 18	535	490	450	451	427
20	1, 2, 4, 6, 7, 8, 10, 12, 14, 15, 17, 18	450	447	429	412	372
21	2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 16, 18	501	477	433	430	427
22	2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18	567	534	491	480	427
23	1, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18	531	498	465	453	372
24	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18	611	587	532	525	427

Table 6
Comparison of mean makespans in benchmark test (ACO).

Prob.	SEA	CCGA	IGA	ACO	Best	Improved rate (%)	CPU time of ACO (seconds)	*S. D. (ACO)
1	437.6	470.4	427	427.1	IGA	0.0	4.1	0.3
2	349.7	369.2	344.5	343.2	ACO	0.4	4	0.6
3	355.2	382	351	347.1	ACO	1.1	4.9	1.7
4	306.2	321.5	307.4	309.4	SEA	−1.0	3	1.6
5	323.7	337.8	309.8	320.4	IGA	−3.4	3	1.2
6	443.8	485.6	427	427.5	IGA	−0.1	7	0.8
7	372.4	385.6	372.7	372	ACO	0.1	3	0.0
8	348.3	373.8	357	344.4	ACO	1.1	4	1.2
9	434.9	474.5	427	427	IGA, ACO	0.0	6	0.0
10	456.5	502.6	431.6	429.2	ACO	0.6	10	1.9
11	378.9	423.8	379.7	368.3	ACO	2.8	9.4	3.7
12	332.8	379.5	323.7	337.7	IGA	−4.3	6.8	3.1
13	469	511.1	442.8	433.2	ACO	2.2	13	4.2
14	402.4	433.4	415.3	385.1	ACO	4.3	8	2.3
15	445.2	493.9	427.4	427.3	ACO	0.0	12	0.5
16	478.8	549.7	449.4	442.2	ACO	1.6	16.5	3.5
17	448.9	496.9	426	414.3	ACO	2.7	18.7	7.6
18	389.6	419.8	373.6	383.7	IGA	−2.7	15.3	3.6
19	508.1	557	471.3	460.4	ACO	2.3	21	4.4
20	453.8	482.7	446.6	421.6	ACO	5.6	15.1	5.0
21	483.2	534	447.8	435.7	ACO	2.7	20.1	3.3
22	548.3	587.5	508.1	484.2	ACO	4.7	30.1	3.9
23	507.5	557.9	477.8	462.8	ACO	3.1	26	5.5
24	602.2	633.3	548.5	531.1	ACO	3.2	40	4.5

∗S. D.: Standard Deviation

7. Conclusions

This paper presents the novel approach to incorporate constructive meta-heuristics to solve the IPPS problem. The ant colony optimization (ACO) is illustrated as an example.

To make the IPPS problem solvable by constructive meta-heuristics, the IPPS problem is transformed to a sequencing-like problem, that is, to select necessary operations for each job and arrange them in a whole sequence. A function which is called “mapping rule” is then used to establish the corresponding schedule. It has been shown that the IPPS problem can be modeled as an integer programming problem.

The proposed model is further analyzed and decomposed into three sub-problems: calculating the search frontier and state transition, establishing search strategy, and developing a schedule calculation function called mapping rule. Then, this paper provides a generic framework which encapsulates the search frontier calculation and state transition, and leaves two interfaces the search strategy and mapping rule to accommodate different constructive heuristic methods. As an example, the shortest processing time first (SPT) rule and ACO strategy are integrated as the search strategy, while a simple and an enhanced mapping rule is constructed.

In this paper, the IPPS is modeled as a single decision-making problem, and the proposed framework provides a uniform solution strategy, it means that the process planning and scheduling functions are truly integrated. The experiments on benchmark problems have shown the effectiveness and high performance of the proposed approaches integrated with an ACO strategy.

With regard to the ACO implementation, it is found that there are still some common problems. The ability of ACO in the IPPS problem field seems limited by the flexibilities of process plans. On one hand, alternatives may distract pheromones deposited on each trail. If the pheromone density cannot be successively strengthened, the significance of a preferable path may fade quickly. In this circumstance, to guide ants intensively searching an area around a promising solution, a large amount of pheromone is needed to be laid every time. Accordingly, there will be a higher chance for the algorithm to fall into local optimal, and loses the ability to explore new areas. On the other hand, if the amount of deposited pheromones is reduced each time and the pheromone evaporation rate is increased, the preferable path cannot be strengthened. The ants will wander on the graph aimlessly in which case even a “local optimal” solution cannot be reached. It is a problem of balancing between the intensification and diversification, which has to be addressed through trial and error. So far, there are not yet systematic approaches or theories to rationalize the parameter setting to address the balancing issue.

Acknowledgment

The work described in this paper is fully supported by a Grant from the Research Grants Council of Hong Kong (Project code HKU 718809E).

Reference

[1] C. Blum, Beam-ACO—Hybridizing ant colony optimization with beam search: An application to open shop scheduling, Comput. Oper. Res. 32 (2005) 1565–1591.
[2] F. Cay, C. Chassapis, An IT view on perspectives of computer aided process planning research, Comput. Ind. 34 (1997) 307–337.
[3] G. Chryssolouris, S. Chan, N.P. Suh, An integrated approach to process planning and scheduling, CIRP Ann. Manuf. Technol. 34 (1985) 413–417.
[4] A. Colorni, M. Dorigo, V. Maniezzo, M. Trubian, Ant system for job-shop scheduling, Belg. J. Oper. Res. Stat. Comput. Sci. 34 (1994) 39–53.
[5] M. Dorigo, V. Maniezzo, A. Colorni, Ant system: Optimization by a colony of cooperating agents, IEEE Trans. Syst. Man Cybern. Part B Cybern. 26 (1996) 29–41.
[6] M. Dorigo, T. Stützle, Ant Colony Optimization, Bradford Company, Scituate, MA, USA, 2004.
[7] H. Gökdağ, A.R. Yildiz, Structural damage detection using modal parameters and particle swarm optimization, Mater. Test. 54 (2012) 416–420.
[8] M. Gen, L. Lin, Multiobjective evolutionary algorithm for manufacturing scheduling problems: State-of-the-art survey, J. Intell. Manuf. 25 (2014) 849–866.
[9] M. Gendreau, J.-Y. Potvin, Metaheuristics in combinatorial optimization, Ann. Oper. Res. 140 (2005) 189–213.
[10] Y. Guo, W. Li, A.R. Mileham, G.W. Owen, Applications of particle swarm optimisation in integrated process planning and scheduling, Robot. Comput. Integr. Manuf. 25 (2009) 280–288.
[11] N. He, D. Zhang, Q. Li, Agent-based hierarchical production planning and scheduling in make-to-order manufacturing system, Int. J. Prod. Econ. 149 (2014) 117–130.
[12] J. Heinonen, F. Pettersson, Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem, Appl. Math. Comput. 187 (2007) 989–998.
[13] K.-L. Huang, C.-J. Liao, Ant colony optimization combined with taboo search for the job shop scheduling problem, Comput. Oper. Res. 35 (2008) 1030–1046.
[14] Y.K. Kim, K. Park, J. Ko, A symbiotic evolutionary algorithm for the integration of process planning and job shop scheduling, Comput. Oper. Res. 30 (2003) 1151–1171.
[15] R. Kumar, M. Tiwari, R. Shankar, Scheduling of flexible manufacturing systems: An ant colony optimization approach, Proc. Inst. Mech. Eng. Part B J. Eng. Manuf. 217 (2003) 1443–1453.
[16] H. Lee, S.-S. Kim, Integration of process planning and scheduling using simulation based genetic algorithms, Int. J. Adv. Manuf. Technol. 18 (2001) 586–590.
[17] C.W. Leung, T.N. Wong, K.L. Mak, R.Y.K. Fung, Integrated process planning and scheduling by an agent-based ant colony optimization, Comput. Ind. Eng. 59 (2010) 166–180.
[18] D. Merkle, M. Middendorf, H. Schmeck, Ant colony optimization for resource-constrained project scheduling, IEEE Trans. Evolut. Comput. 6 (2002) 333–346.
[19] N. Morad, A. Zalzala, Genetic algorithms in integrated process planning and scheduling, J. Intell. Manuf. 10 (1999) 169–179.
[20] M.A. Potter, K.A. De Jong, A cooperative coevolutionary approach to function optimization, in: Proceedings of the Third Conference on Parallel Problem Solving from Nature (PPSN). Proceedings of the International Conference on Evolutionary Computation, 866, 1994, pp. 249–257.
[21] L.H. Qiao, S.P. Lv, An improved genetic algorithm for integrated process planning and scheduling, Int. J. Adv. Manuf. Technol. 58 (2012) 727–740.
[22] A. Rossi, Flexible job shop scheduling with sequence-dependent setup and transportation times by ant colony with reinforced pheromone relationships, Int. J. Prod. Econ. 153 (2014) 253–267.
[23] C. Saygin, S. Kilic, Integrating flexible process plans with scheduling in flexible manufacturing systems, Int. J. Adv. Manuf. Technol. 15 (1999) 268–280.
[24] X. Shao, X. Li, L. Gao, C. Zhang, Integration of process planning and scheduling—A modified genetic algorithm-based approach, Comput. Oper. Res. 36 (2009) 2082–2096.
[25] W. Shen, L. Wang, Q. Hao, Agent-based distributed manufacturing process planning and scheduling: A state-of-the-art survey, IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. 36 (2006) 563–577.
[26] X.-N. Shen, X. Yao, Mathematical modeling and multi-objective evolutionary algorithms applied to dynamic flexible job shop scheduling problems, Inf. Sci. 298 (2015) 198–224.

[27] V. T'kindt, N. Monmarché, F. Tercinet, D. Laügt, An ant colony optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem, Eur. J. Oper. Res. 142 (2002) 250–257.
[28] T. Wong, S. Zhang, G. Wang, L. Zhang, Integrated process planning and scheduling - Multi-agent system with two-stage ant colony optimisation algorithm, Int. J. Prod. Res. 50 (2012) 6188–6201.
[29] T.N. Wong, C.W. Leung, K.L. Mak, R.Y.K. Fung, Dynamic shopfloor scheduling in multi-agent manufacturing systems, Expert Syst. Appl. 31 (2006) 486–494.
[30] T.N. Wong, C.W. Leungy, K.L. Mak, R.Y.K. Fung, Integrated process planning and scheduling/rescheduling - An agent-based approach, Int. J. Prod. Res. 44 (2006) 3627–3655.
[31] L.-N. Xing, Y.-W. Chen, P. Wang, Q.-S. Zhao, J. Xiong, A knowledge-based ant colony optimization for flexible job shop scheduling problems, Appl. Soft Comput. 10 (2010) 888–896.
[32] X. Xu, Integrating advanced computer-aided design, manufacturing, and numerical control: principles and implementations, Information Science Reference Hershey (2009).
[33] B. Yagmahan, M.M. Yenisey, Ant colony optimization for multi-objective flow shop scheduling problem, Comput. Ind. Eng. 54 (2008) 411–420.
[34] A.R. Yildiz, A comparative study of population-based optimization algorithms for turning operations, Inf. Sci. 210 (2012) 81–88.
[35] A.R. Yildiz, Comparison of evolutionary-based optimization algorithms for structural design optimization, Eng. Appl. Artif. Intell. 26 (2013) 327–333.
[36] A.R. Yildiz, A new hybrid particle swarm optimization approach for structural design optimization in the automotive industry, Proc. Inst. Mech. Eng. Part D J. Automob. Eng. 226 (2012) 1340–1351.
[37] A.R. Yildiz, Optimization of multi-pass turning operations using hybrid teaching learning-based approach, Int. J. Adv. Manuf. Technol. 66 (2013) 1319–1326.
[38] A.R. Yildiz, K.N. Solanki, Multi-objective optimization of vehicle crashworthiness using a new particle swarm based approach, Int. J. Adv. Manuf. Technol. 59 (2012) 367–376.
[39] H.C. Zhang, L. Alting, Computerized Manufacturing Process Planning Systems, Chapman & Hall, Ltd., 1994.
[40] L. Zhang, T. Wong, R.Y. Fung, A multi-agent system to support heuristic-based dynamic manufacturing rescheduling, Intell. Decis. Technol. 7 (2013) 197–211.
[41] L. Zhang, T.N. Wong, R.K. Fung, A multi-agent system for dynamic integrated process planning and scheduling using heuristics, in: G. Jezic, M. Kusek, N.-T. Nguyen, R. Howlett (Eds.), Agent and Multi-Agent Systems: Technologies and Applications, Springer, Berlin Heidelberg, 2012, pp. 309–318.
[42] S. Zhang, T. Wong, L. Zhang, S. Wan, A two-stage approach based on ant colony optimization algorithm for integrated process planning and scheduling, in: Proceedings of the Forty-first International Conference on Computers and Industrial Engineering, LA, US, 2011, pp. 804–809.
[43] W. Zhang, M. Gen, Process planning and scheduling in distributed manufacturing system using multiobjective genetic algorithm, IEEJ Trans. Electr. Electron. Eng. 5 (2010) 62–72.
[44] W. Zhang, M. Gen, J. Jo, Hybrid sampling strategy-based multiobjective evolutionary algorithm for process planning and scheduling problem, J. Intell. Manuf. 25 (2014) 881–897.