A Novel Search Framework for Multi-Stage Process Scheduling with Tight Due Dates

Yaohua He
The Logistics Institute – Asia Pacific, National University of Singapore/Georgia Institute of Technology, Singapore 119613

Chemical and Biomolecular Engineering Dept., Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong, P.R. China

Chi-Wai Hui
Chemical and Biomolecular Engineering Dept., Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong, P.R. China

DOI 10.1002/aic.12134
Published online December 30, 2009 in Wiley InterScience (www.interscience.wiley.com).

This article improves the original genetic algorithm developed by He and Hui (Chem Eng Sci. 2007; 62:1504–1527) and proposes a novel global search framework (GSF) for the large-size multi-stage process scheduling problems. This work first constructs a comprehensive set of position selection rules according to the impact factors analysis presented by He and Hui (in this publication in 2007), and then selects suitable rules for schedule synthesis. In coping with infeasibility emerging during the search, a penalty function is adopted to force the algorithm to approach the feasible solutions. The large-size problems with tight due dates are challenging to the current solution techniques. Inspired by the gradient used in numerical analysis, we treat the deviation existing among the computational tests of the algorithm as evolutionary gradient. Based on this concept, a GSF is laid out to fully utilize the search ability of the current algorithm. Numerical experiments indicate that the proposed search framework solves such problems with satisfactory solutions. © 2009 American Institute of Chemical Engineers *AIChE J,* 56: 2103–2121, 2010

Keywords: hybrid flow shop, multi-stage process scheduling, genetic algorithm, heuristic rules, evolutionary gradient, global search framework

Introduction

A hybrid flow shop (HFS),[1] also called flexible flow shop, can be regarded as a generalized flow shop with several processing stages. In each stage, there are a number of unrelated parallel machines for processing operation. All jobs to be processed in the HFS need to go through all the stages in the same sequence. A job can be processed on one of the machines at a stage. It is a special case of HFS that machines at a given stage are identical. If there is only one machine at every stage, the HFS becomes a regular flow shop. If there is only one stage, then HFS becomes a workshop with parallel machines. HFS is common in continuous process industries such as chemical industry and steel making industry. But in the process industry, HFS is usually called ‘‘a multi-stage multi-product batch plant’’.

Multi-stage multi-product scheduling problems (MMSP) in batch plants are very difficult but routine problems which are much more complex than single-stage multi-product scheduling problems (SMSP) with parallel units. Such

Additional Supporting Information may be found in the online version of this article.
Correspondence concerning this article should be addressed to Y. He at YaohuaHe2009@gmail.com.

problems have been always attractive to the process scheduling researchers. There is a large body of literature on MMSP.

Pinto and Grossmann[2] presented a continuous-time mixed-integer linear programming (MILP) model for MMSP. They used the concept of parallel time coordinates for units and tasks. Pinto and Grossmann[3] still proposed an alternative model in which the pre-ordering of orders was imposed explicitly, by applying a representation of the time slots for the units. This resulted in a significant reduction in the computational time.

Hui et al.[4] presented a general continuous-time MILP model for MMSP. The requirement of binary decision variables was considerably reduced by the application of tri-index variables to represent order sequencing and changeover. The main advantage of this model compared to the other proposed formulations so far was the significant reduction in the number of binary variables.

Mendez et al.[5] proposed another sequence-based MILP formulation with a slightly different definition of three-index binary variables (batch, batch, and stage). Instead of sequencing adjacent batches, they used indirect relative sequencing of all batches. Furthermore, they addressed limited discrete renewable resources by using additional four-index binary variables (batch, stage, batch, and stage). One important feature of their model is that it uses fewer binary variables. Unfortunately, they did not bring about any performance improvements and they did not compare their model with that of Hui et al.[4]

Subsequently, Gupta and Karimi[6,7] developed several improved sequence-based MILP formulations using a sequencing approach similar to that of Hui et al.[4] Their formulations used fewer binary variables and constraints, and solved faster. In some cases, they even obtained better solutions.

Castro and Grossmann[8] developed a continuous-time MILP model with multiple time-grids for MMSP. They considered three minimization objectives (total cost, total earliness, and makespan) and compared their approach with other approaches in the literature. Although their model used the same four-index binary variables as those of Pinto and Grossmann,[2] they argued that their model required fewer slots and hence was tighter and faster than the earlier model. Castro et al.[9] later presented two new multiple-time-grid continuous-time formulations for SMSP and MMSP, where equipment units were subject to sequence-dependent changeovers, and product orders were subject to both release and due dates, where the objective was the minimization of total cost, total earliness, or makespan. For makespan minimization, the multiple-time-grid, continuous-time formulation by Castro et al.[9] was not the best efficient mathematical programming method. The constraint programming (CP) method and the discrete-time formulation were found to be the best mathematical programming methods. However, these two methods have the disadvantage of considering integer data and show a rapid decrease in performance with an increase in problem complexity.

Liu and Karimi[10] constructed and compared several novel MILP formulations for the scheduling of multi-stage batch plants with identical parallel units. In contrast to the existing work, they increased solution efficiency by considering each stage as a block of multiple identical units, thereby eliminating numerous binary variables for assigning batches to specific units. A novel formulation using an adjacent pair-wise sequencing approach proved superior to slot-based formulations. Furthermore, they developed heuristic variations of their proposed formulations to address moderate-size problems. A novel heuristic strategy inspired from list scheduling algorithms seems to be efficient for moderate-size problems and scales well with problem size.

Liu and Karimi[11] constructed, analyzed, and rigorously compared a variety of novel MILP formulations using unit-slots, stage-slots, process-slots, a variety of slot arrangements, and sequence-modeling techniques, four-index and three-index binary variables, etc. While two of four-index models are an order of magnitude faster than existing models on 22 test problems of varying sizes, no single model performs consistently the best for all problems. They suggested that the best strategy for solving difficult scheduling problems might be to use a set of competitive models in parallel and terminate them all when the desired solution available. They also developed several heuristic models based on their formulations. Even a heuristic based on an inferior model can surpass others based on superior models.

All this work assumed unlimited intermediate storage (UIS) and unlimited wait policy. All of the above methods are MILP with focus on small-size problems. When the problem size increases linearly, the computational time of MILP will increase exponentially. It is very difficult for MILP to solve large-size problems. Although some researchers[12,13] claimed that their MILP models were suitable for large-size problems, the solutions to large-size problems were far from the optimum, even if the algorithm ran for very long time.

In addition to the above efforts that used purely MILP-based methods, Harjunkoski and Grossmann[14] and Maravelias[15] proposed decomposition methods for single- and multi-stage problems in which they decomposed the original problem into two sub-problems (an assignment sub-problem and a sequencing sub-problem). They used different methods for these sub-problems, and generated and imposed cuts at each iteration. While these methods are still heuristic, as the cuts may lead to suboptimal solutions, they offer promise due to their ability to solve moderately larger problems.

If a ‘‘good enough’’ solution is the goal, then heuristic models and meta-heuristics are the best choice at present. Liu and Karimi[11] stated that ‘‘The meta-heuristics are indeed attractive, as they are quite simple to apply to many problems and often obtain good solutions within reasonable times. However, all require some ad hoc or experiment-based selection of arbitrary search parameters in one way or other.’’

Compared to mathematical programming methods, meta-heuristic methods are more suitable for large-size problems. Wang et al.[16] proposed a genetic algorithm (GA) for the online-scheduling of a multi-product polymer batch plant. It was seen that GA outperformed mathematical programming. Moreover, the modeling effort for GA is considerably lower than that for mathematic programming. The ‘‘modeling effort’’ is mainly referred to the effort needed to construct the models of the combinatorial problems. For example, in solving the traveling salesman problem (TSP) by MILP, the constraints to avoid sub-tours contain a large number of equations. Scheduling problems are usually typical combinatorial problems which are often NP-hard (nondeterministic polynomial time, NP) like TSP. The MILP or MINLP models for scheduling problems need quite a number of constraints to ensure feasible solutions in solving the models.

In fact, in the community of discrete production scheduling, a lot of publications on HFS scheduling can be found. The scheduling of HFS is a complex combinatorial problem encountered in many real world applications. Given its importance and complexity, the HFS problem has been intensively studied. Most recently, Ruiz and Vazquez-Rodriguez[17] have presented a literature review on exact, heuristic and meta-heuristic methods proposed for the solutions of HFS scheduling problems. It can be found that GA, simulated annealing (SA) and tabu search (TS) are the most widely used meta-heuristics in HFS scheduling or other types of scheduling.

Chen et al.[18] developed an approach for applying GA to continuous flow shop scheduling problems (FSSP). In Ruiz and Maroto,[19] a GA was employed to minimize makespan on an m-stage problem with unrelated parallel machines, sequence dependent setup times and machine eligibility. They adopted a restart strategy to overcome the premature convergence of GA.

Tandon et al.[20] presented a solution methodology to obtain near-optimal solutions of the scheduling problems of multiple products on unrelated parallel units using an SA algorithm. Near-optimal solutions were obtained for problems with 20–100 products and 3–11 units. Low[21] proposed an SA method for HFS with total flow time minimization. He also considered sequence dependent removal times and sequence independent setup times. Allahverdi and Al-Anzi[22] presented several heuristics and an SA method for an m-stage HFS.

An m-stage problem with group scheduling was approached by Logendran et al.[23] with TS. Another application of TS is for the regular HFS with the sum of the weighted completion times criterion and limited buffers by Wang and Tang.[24]

We note that the original problem data of MMSP[2,4] were tailored for their model with loose due dates. Even after we doubled the problem size, but not change the due dates, we still obtained good feasible solutions.[25] And yet, after shortening the due dates, which lead to tardy orders even at the optimum, we saw infeasible solutions in minimizing earliness related objectives despite the relaxation of soft due dates. Theoretically, with the relaxation of soft due dates, feasible solutions must exist, because, at least, the solution for setting the soft due dates is feasible. If feasible solutions under soft due dates are not able to achieve, then the weak search ability of the algorithm is the sole reason. So the scheduling problem with tight due dates, which often appears in practice, is challenging to the solution methods.

In our previous work,[25] we have developed a GA for the large-size MMSP. First, unit selection rules and the active scheduling technique were subtly combined into the GA; secondly, a penalty method was used to deal with the constraints of the forbidden changeovers between orders and the forbidden processes on some units; thirdly, the concept of ‘‘soft due date’’ was adopted to tackle the problem with tardy orders at the optimality. Compared to MILP, our algorithm performs better both in terms of efficiency and computational effort, as well as in solution quality. Even so, in practical applications of the meta-heuristics like GA, the users usually have to tune the algorithm parameters according to the problem conditions and carry out a number of tests to screen out the best solution for real use. In general, the users hope to have lower variance of solutions in the tests, which may imply the algorithm has higher search performance. But the search ability of an algorithm depends on the prevailing problem conditions. As long as the solution variance exists, all the better solutions via the algorithm are still possible. So, how to fully utilize the search ability of the current algorithm until the average deviation turns to be null is what to be considered in this article.

In this article, we first improved the GA from two aspects: (1) a comprehensive set of position selection rules are systemically summarized via the impact factors analysis method,[26] and suitable rules are selected for schedule synthesis (i.e., decoding procedure of GA); (2) for the infeasible solutions appearing during the search minimizing the earliness related objectives, a penalty function is used, hence leads the algorithm to find feasible solutions. Computational experiments reveal that the improved GA (IGA) increases its search ability to solve the large-size instances in our previous work. However, the large-size instances with tight due dates (with tardy orders even at the optimality) are still very difficult to be solved to feasibility. For such difficult problems, a global search framework (GSF) is then designed based on the idea of evolutionary gradient. To test the solution ability of the IGA and the novel search framework, they are applied to much larger general MMSP generated randomly. The algorithms in this work were implemented in C language and the computational tests were run on a PC with 1500 MHz CPU and 768M memory.

Problem Description of MMSP

An MMSP is defined as follows. A batch plant has a total of M_T processing units (forming a set of units, U) and has received N customer orders (forming a set of orders, O). Each order, involving a single product and requiring a certain number of processing stages (M), has a predetermined due date and a release time (the earliest start time for production). There is UIS between stages. A fixed number of unrelated parallel units (a subset of U) are available at each processing stage to process the orders. Assume the number of parallel units in stage k is m_k, $k = 1, 2, \ldots, M$. So $\sum_{k=1}^{M} m_k = M_T$. Each unit maybe also has a release time (the earliest available time for production). In each stage, only one unit processes an order. No unit coexists in more than one processing stage. The processing time of an order over a unit depends on the nature of the order and on the type of the unit. Not all the orders can be processed by any unit (forbidden processes exist, in these cases let the process times be represented by dashes). At each stage, a changeover time is required when a unit change from one order to another. Changeover times are sequence dependent. Some changeovers are forbidden (let the changeover times represented by dashes). A unit setup time is required for every order changeover, which is either sequence or unit dependent. Forbidden changeovers and forbidden processes in the problem are called CP constraints. The scheduling objective is to minimize the total flow time, total tardiness or total earliness of the schedule.

To illustrate the tri-index continuous-time MILP model for MMSP in a batch plant, Pinto and Grossmann[2] presented small-size MMSP which were later solved by a bi-index model proposed by Hui et al.[4] In this study, the small-size MMSP are extended to large-size MMSP which are difficult for the current MILP models to solve to optimality. In the batch plant, there are 25 units available to process up to 24 customer orders ($M_T = 25$, $N = 24$). Each order involves five stages ($M = 5$). Units 1–6 are the parallel processing units in Stage 1, Units 7–9 in Stage 2, Units 10–19 in Stage 3, Units 20–22 in Stage 4, and Units 23–25 in Stage 5. The unit-dependent process times (p_{ju}, $j = 1, 2, \dots, 24$, $u = 1, 2, \dots, 25$), sequence-dependent changeover times (c_{ij}, i and $j = 1, 2, \dots, 24$), unit setup times (ut_u, $u = 1, 2, \dots, 25$), order due date (d_j, $j = 1, 2, \dots, 24$), order release times (or_j, $j = 1, 2, \dots, 24$) and unit release times (ur_u, $u = 1, 2, \dots, 25$) are presented in Supporting Information Tables A-1 and A-2 in Appendix A. The dashes in these two tables mean forbidden processes or changeovers. Example 1 is first used to illustrate the proposed approach. Examples 2 and 3 are then to be used for case study. Examples 1, 2, and 3 here are Examples 1, 3, and 4 in He and Hui.[25]

Example 1

The process times in Table A-1, the changeover times and due dates in Table A-2 are used in this example. All unit setup times, order release times and unit release times in Example 1 are set to be zero. The small size (not more than 12 orders) instances of Example 1 were studied by pinto and Grossmann[2] and Hui et al.[4]

Example 2

The unit setup times, the unit release times in Table A-1 and the order release times in Table A-2 are used in this example. All the other data are the same as Example 1.

Example 3

The due dates of the former 6 orders the are changed from (510, 500, 520, 530, 540, 530) to (50, 100, 200, 300, 400, 500). All the other data are the same as Example 2.

It can be noted that the MMSP is an expansion of the FSSP, or called the flexible or HFS scheduling problem in some literature.[17,27] In regular FSSP, there is only one processing unit at every stage of production. The difference of MMSP is that there are several parallel process units available for operations in every stage. For every stage, assignment of orders to the parallel units in this stage is a similar SMSP. In other words, the flow shop plant and the single-stage plant with parallel units are special cases of the multi-stage plants.

The performance criteria to evaluate the meta-heuristic algorithms are the same as described in He and Hui,[25] the relative deviation (RD) and the computational time. Considering different scheduling objectives, the RD is calculated with respect to the best (or optimal) solution obtained up to now:

$$\textit{dev. from best}\,(\%) = 100 \times \frac{\text{objective value} - \text{best objective value}}{\text{best objective value}}. \qquad (1)$$

Solution by MILP

The solution procedure of the MILP model proposed by Hui et al.[4] is conducted as follows: establish the MILP model first, including objective function and constraints, and then the model is formulated by GAMS[28] and solved by OSL on the PC with 1500 MHz CPU and 768MB memory. The scheduling objective is to minimize total flow time, total tardiness, or total earliness. The sum of completion times is in the literature[29] often referred to as total flow time F:

$$\min. \quad F = \sum_{j=1}^{N} C_j, \qquad (2)$$

where C_j is the completion time of order j; for the total tardiness, the objective function is:

$$\min. \quad T = \sum_{j=1}^{N} T_j, \qquad (3)$$

where $T_j = \max\{C_j - d_j, 0\}$ is the tardiness of order j; for the *total earliness*, the objective function is:

$$\min. \quad E = \sum_{j=1}^{N} E_j, \qquad (4)$$

where $E_j = \max\{d_j - C_j, 0\}$ is the earliness of order.

Pinto and Grossmann[2] proposed an objective function for minimization of the total process time, and Hui et al.[4] adopted this objective function. In our study, a new objective function is proposed. The process time of order j through all stages is $PR_j = C_j - S_j$; $\sum_{j=1}^{N}(C_j - S_j)$ is the total process time, where S_j is the start time of order j. Assume that all orders can be completed before their due dates, so the earliness of order j is $E_j = d_j - C_j$; $E_j + PR_j = (d_j - C_j) + (C_j - S_j) = d_j - S_j$. We define $(d_j - S_j)$ as the generalized process time of order j. Hence, the total generalized process time, simply as total process time is:

$$\min. \quad PT = \sum_{j=1}^{N} \left(d_j - S_j\right). \qquad (5)$$

This objective function is the compound of ‘‘total process time’’ and ‘‘earliness’’ and has the same function as that proposed by Pinto and Grossmann[2]: although it does not guarantee minimization of total earliness, it has the advantage of minimizing the total in-process time. This would reflect minimum intermediate storage requirements.

From the results of Example 1 solved by the MILP model developed by Hui et al.[4] for minimizing PT (see Table 2), also from our observation of the computational process of the MILP model, the limitation of MILP for the problem with highly combinatorial nature can be evidently seen:

(1) For small-size instances, MILP can get the optimal solution of the instance within short time. For example, the optimal solution for the 5-order instance has been obtained within 541 iterations taking only 0.43s.

(2) For large-size instances, MILP cannot get the optimal solution within acceptable time. In Table 2, for the 8-order instance onwards, the optimal solution cannot be achieved within 100,000 iterations. Indeed, the 100,000 iterations are acceptable. However, for the large-size instances, even we increase the iterations up to 1,000,000 or 2,000,000 (over 2 h), the solutions are improved very little.

(3) We also observed that the more CP constraints in the problem, the more rapidly the MILP method finds the solution. The reason for this is that the CP constraints cut some solution space for MILP.

In our previous work,[25] we provided four Gantt charts of the schedules obtained by the MILP. From the Gantt charts, it is seen that the optimal schedule corresponds to a uniform Gantt chart without idle times. On the contrary, a nonoptimal schedule may witness a Gantt chart with big idle times.

For the MMSP solved in this paper, Pinto and Grossmann[2] proposed an original MILP model, later Hui et al., Gupta and Karimi,[6,7] and Liu and Karimi[11] presented their modified MILP models. All these works focused small problems, and the later improvements were marginal, not substantial, especially for large-size problems. For large-size instances, MILP may be able to get a feasible solution, but it is far from the optimal solution. All in all, MILP is not suitable to solve large-size MMSP. So the MILP model by Hui et al.,[4] may not be so efficient as the other later models, but it is representative.

The Improved Genetic Algorithm

In the original GA (OGA)[25] to solve the large-size MMSP, a set of solutions to the problem are represented by chromosomes to be evolved through genetic mechanism. A chromosome consists of order sequences corresponding to the processing stages. These order sequences are then assigned to processing units according to assignment strategies such as forward or backward assignment, the active scheduling technique or similar technique, and some unit selection rules. All these measures greatly increase the search speed.

In this article, the basic structure and components of OGA are still adopted in the IGA, the improvement focuses on two aspects:

(1) Instead of using a unit selection rule chosen from a set of rules to determine a unit for order assignment, a position selection rule is applied for selecting a suitable position among all the possible positions for order assignment. The position selection rule is screened out from a set of position selection rules which are derived from the impact factors analysis method.[26]

(2) Penalty to the infeasible schedules appearing in the process of search. For the minimization of earliness related objectives, due to the backward assignment method used, some operations of an order may not be arranged within the time horizon ranging from its release time to its soft due date. These orders will have a start time $S_j < or_j$. If $or_j = 0$, then negative start times arise in the schedule. Thus, the schedule yielded is infeasible. Infeasible schedules are tolerable in the course of search, but not acceptable for the final solution. In order to obtain feasible solutions in every computational test, a penalty objective function is adopted.

Position selection rules

Before understanding position selection, the concepts of idle time and onward position should be clear. In fact, these two concepts have been introduced in active scheduling in our previous work.[25] In the forward strategy, from the unit release time to the final operation assigned so far on a unit, if any time interval possible to assign another operation exists, this time interval is an idle time. The time after the final operation assigned so far is the onward position. Similarly, we can define the idle times with respect to due dates in the backward strategy.

At each stage of the MMSP, there are several parallel units available for an order to assign to. If we do not consider the idle times between the assigned operations, then assigning orders to the parallel units onward is a sub-problem of SMSP, and a unit selection rule can be applied. For SMSP, based on the impact factors analysis, we have summarized seven unit selection rules for minimizing the makespan related objectives, such as the makespan, total flow time and total tardiness,[26] where Rule 6 is:

Rule 6: Assign the order on the unit that makes the sum of the order's changeover time and process time on the unit be the shortest.

For MMSP, considering the unit setup times, Rule 6 should be changed into:

Rule 6: Assign the order to the unit that makes the sum of the order's changeover time, unit setup time and process time on the unit to be the shortest.

Rule 6 can be used in our algorithms for MMSP no matter what objective is selected, because this rule does not involve the factor of when to start an order. In OGA, Rule 6 is used first to select a unit for a new coming order; and then "first available idle time" is used to select an idle time on the unit to assign the order.

For MMSP to minimize the makespan-related objectives by forward assignment strategy, if there are a large number orders to be processed on a small number of parallel units, then for each stage except Stage 1, there may be several idle times available for a new coming order. Selecting the "first available idle time" is one choice, but not the only choice. For example, the "earliest completion time" is another choice. Actually, if the idle times and the onward positions on all the parallel units in the current stage are referred to as "positions", then a set of position selection rules can be summarized as the criteria to assign an order, as shown in Table 1.

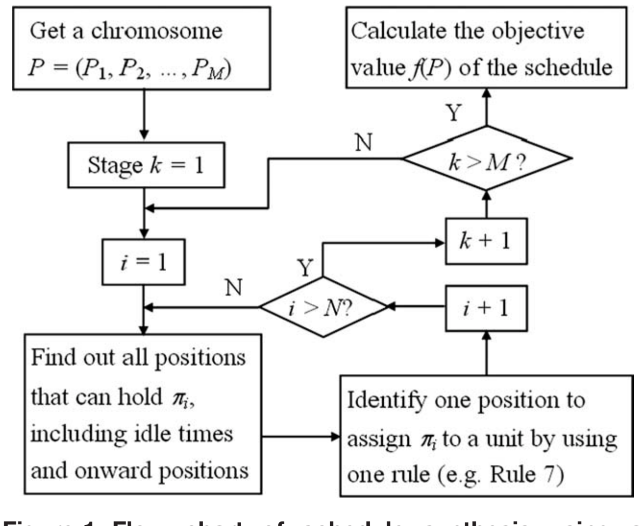
Figure 1 shows the flow chart of synthesizing a chromosome into schedule according to the forward assignment strategy, active scheduling technique and one position selection rule. The procedure of schedule synthesis with backward assignment strategy is similar to Figure 1.

Figure 2 shows us the illustration of position selection in forward strategy. To synthesize a chromosome into a schedule, two kinds of time pointers are originally adopted in our algorithms. One is the kind of the order time pointers, JP_i,

Table 1. Seven Rules for Minimization of the Makespan-Related Objectives in MMSP

Rule	Detailed Content	Shortened Form
Rule 1	Assign the order on the position that makes the order's possible start time be as early as possible, that is, assign the order on the first available position	First available position, FAP (earliest possible start time, PsT)
Rule 2	Assign the order on the position that makes the order's changeover time on the position be the shortest	shortest changeover time, SCT
Rule 3	Assign the order on the position that makes the sum of the order's unit setup time and process time on the position be the shortest	shortest unit setup time + process time, SUPT
Rule 4	Assign the order on the position that makes the order's start time (PsT + CT) be as early as possible	earliest start time, EST
Rule 5	Assign the order on the position that makes the sum of the order's possible start time, unit setup time and process time on the position be the shortest	shortest possible start time + unit setup time + process time, SPsU*PT*
Rule 6	Assign the order on the position that makes the sum of the order's changeover time, unit setup time and process time on the position be the shortest	shortest changeover time + unit setup time + process time, SCUPT
Rule 7	Assign the order on the position that makes the order's completion time be as early as possible	earliest completion time, ECT

$i = 1, 2, 3, \ldots, N$, the other is the kind of unit time pointers, UP_u, $u = 1, 2, 3, \ldots, M_T$. In the forward strategy, the initial value of JP_i is the order release time or_i, and the initial value of UP_u is the unit release time ur_u. Assume that all operations of all the orders in stage (k–1) have been assigned to the units in this stage, and some operations in stage k have been assigned, as shown in Figure 2a. The operation of order i is to be assigned to one of the parallel units in stage k. JP_i is the current completion time of the operation of the order i in stage (k–1), and UP_u is the completion time of the last operation on unit u (if one or more assigned operations exist; else $UP_u = ur_u$). If $JP_i > UP_u$, only the onward position on unit u can admit the operation of order i, hence, on the units u_{k1} and u_{k4}, only the onward positions after the JP_i can be used. If $JP_i \leq UP_u$, besides the onward position on unit u, there may be idle times that can admit the operation of order i. Hence, on the units u_{k2} and u_{k3}, idle times should be considered to assign the operation of order i in this stage. As a result, there are five available positions to assign order i in sage k, as shown in Figure 2b.



Get a chromosome $P = (P_1, P_2, \ldots, P_M)$
Calculate the objective value $f(P)$ of the schedule
Y
N
Stage $k = 1$
$k > M$?
$k + 1$
$i = 1$
Y
N
$i > N$?
$i + 1$
Find out all positions that can hold π_i, including idle times and onward positions
Identify one position to assign π_i to a unit by using one rule (e.g. Rule 7)

Figure 1. Flow chart of schedule synthesis using a position selection rule in forward strategy.

Which position should be selected? Just use one of the position selection rules in Table 1. Rule 1 (first available position) is classically used. However, to choose an appropriate rule, simulation experiments are needed. Through simulation experiments on all the rules listed in Table 1, Rule 7 is found to perform best in solving the investigated problems.

In backward strategy for earliness related objectives, the position selection rules can be changed correspondingly based on the rules in Table 1. For Example, Rule 7 is changed as:

Rule 8: Assign the order on the position that makes the order's start time be as near as possible to its due date.

Similarly, through simulation experiments on all the rules for backward assignment, Rule 8 is found to be the best one in solving the investigated problems.

Two sample schedules of Example 1

According to different scheduling objectives, the sample chromosome in Figure 3 can be synthesized into different schedules by using the data for Example 1 in Appendix A.

For the minimization of the total flow time F (see Eq. 2), the sample chromosome in Figure 3 can be synthesized into a schedule according to the forward assignment strategy, active scheduling technique and Rule 7 (in Table 1). Operations in the sequence $P_1 = (1, 9, 2, 6, 5, 8, 4, 7, 3, 10)$, are assigned to the units u1–u6 in Stage 1 only according to Rule 7; operations in the sequence $P_2 = (6, 5, 7, 3, 8, 2, 9, 1, 10, 4)$, are assigned to the units $u7$–$u9$ in Stage 2 according to the active scheduling technique and Rule 7; and so on, until all operations in the five stages are assigned to the corresponding units. Scheduling in Stage 1 is just a subproblem of SMSP, assigning the orders forward according to Rule 7. From Stage 2 onward, orders are forward assigned to the units according to and the active scheduling technique and Rule 7. As a result, a schedule is obtained with a total flow time $F = 1136.60$, which is much less than the original $F = 1414.90$ in He and Hui.[25]

For the minimization of the total PT (see Eq. 5), the sample chromosome in Figure 3 can be synthesized into a schedule according to the following requirements: (1) backward assignment; (2) technique (similar to active scheduling) to reduce

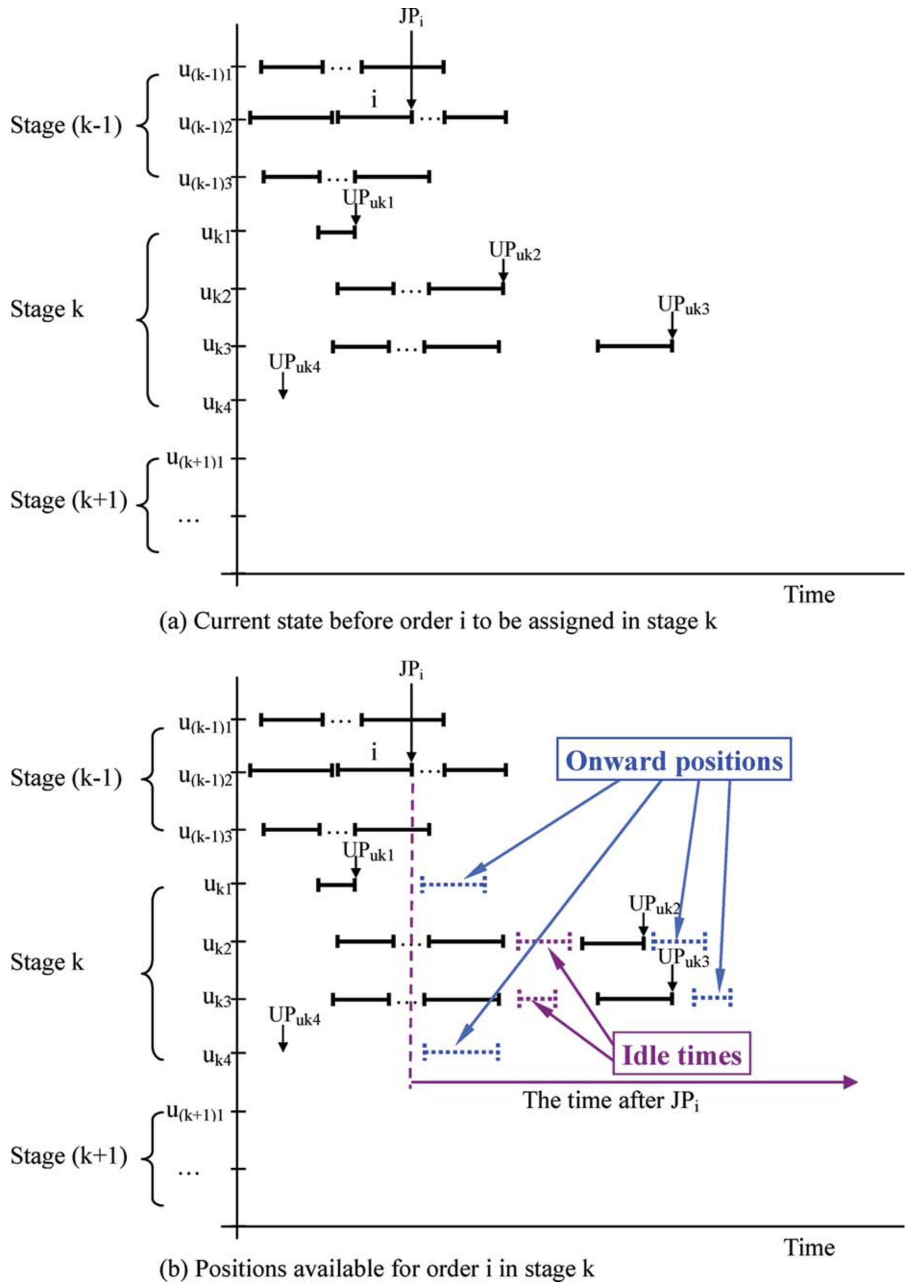


Figure 2. Illustration of position selection.

[Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

the idle times of the units and intermediate storage time of the semi-finished products; (3) for each stage, choose a appropriate position to assign an order according to Rule 8. Consequently, a schedule is obtained with $PT = 819.20$, which is much less than the original $PT = 1232.90$ in He and Hui.[25]

A penalty method to the infeasible schedules

As described in He and Hui,[25] to minimize PT for the problems with tardy orders at the optimum, first minimize T and set the soft due dates of the orders d'_j, $j = 1, 2, \ldots, N$;

(Stage 1)	(Stage 2)	(Stage 3)	(Stage 4)	(Stage 5)
1 9 2 6 5 8 4 7 3 10	6 5 7 3 8 2 9 1 10 4	10 3 9 4 8 5 7 2 6 1	7 4 8 5 9 3 10 2 6 1	9 3 7 1 6 2 10 5 4 8

Figure 3. A sample chromosome for the 5 × 10 ($M = 5$, $N = 10$) instance of Example 1.

Table 2. Best Results of Example 1 by IGA, OGA, and MILP (min. *PT*)

	IGA/OGA				MILP				
$N \times M$	*popsize*	t*	CPU (s)	Best *PT*	*PT*	Iterations	CPU (s)	*Diff1*†	*Diff2*‡
5 × 5	200/200	5/10	0.055/<1	499.40/499.40	499.40	541	0.43	0.00	0.00
8 × 5	200/200	27/19	0.77/1.00	730.90/732.90	736.90	100000	99.56	0.55	0.27
10 × 5	300/300	43/100	1.98/5.00	863.90/876.40	909.30	100000	119.50	4.41	1.45
12 × 5	400/300	60/77	6.32/6.00	1031.90/1038.90	1184.70	100000	125.02	14.03	0.68
16 × 5	600/600	64/161	15.71/14.00	1763.80/1788.80	2048.60	100000	202.50	14.46	1.42
20 × 5	600/600	135/194	35.11/26.00	2413.50/2456.80	2982.50	100000	227.17	21.40	1.79
24 × 5	600/600	178/263	61.00/40.00	3024.50/3239.30	3912.00	100000	374.72	20.77	7.10

*t = iterations in IGA or OGA.
†*Diff1* (%) =100(objective value by MILP – objective value by OGA)/(objective value by OGA).
‡*Diff2* (%) = 100(objective value by OGA – objective value by IGA)/(objective value by IGA).

and then minimize *PT* using the modified objective function below:

$$f(P) = PT = \sum_{j=1}^{N} (d'_j - S_j). \tag{6}$$

When minimizing the earliness-related objectives under the soft due dates, in some computational tests, infeasible schedules may be yielded at the end of the computation. Soft due dates which result from the minimization of the total tardiness T restrict the completion times of the orders under the backward assignment strategy. When an arbitrary chromosome is synthesized into a schedule according to the backward assignment, an order may not be assigned within the time horizon ranging from its release time to its soft due date, hence the schedule obtained is infeasible. An infeasible schedule by the backward assignment strategy is defined as: if in a schedule synthesized there is any order j whose start time $S_j < or_j$ (or_j may be zero), then this is an infeasible schedule.

However, we can rigorously state that under the soft due dates, there must be one or more feasible solutions (schedules) when minimizing the earliness-related objectives with the backward assignment strategy, because, at least, the solution for setting the soft due dates through minimizing tardiness T is a feasible one. In case that enough computational tests are carried out and no feasible solution obtained yet, the search ability of the algorithm should be doubted.

To get feasible solutions in every computational test, an alternative method as follows can be adopted: in the course of computing, if an infeasible schedule from chromosome P comes forth, give its objective value a penalty:

$$f(P) = PT = w\sum_{j=1}^{N} (d'_j - S_j), \tag{7}$$

where w is a coefficient, $w > 1$ (for infeasible schedules), or $w = 1$ (for feasible schedules). We note that such penalty method makes the search time longer in solving large-size instances.

One may argue that the introduction of penalty weight w in Eq. 7 is "a very minor thing". But we should note the impact of this minor issue. Imagine that the existing technique is the sole reliance to solve the difficult problem. Theoretically, we can prove the existence of feasible solutions. However, the feasible solutions are practically intractable by the existing technique. At this time, the introduction of penalty weight w in the objective function allows the existing technique to obtain "good and feasible" solutions. We think that this minor change is significant. The experience is also valuable, which let us understand that infeasible solutions encountered in the search process constitute a bridge to find good feasible solutions. In branch and bound methods, infeasible solutions are usually cut off. But in the meta-heuristic, infeasible solutions are utilized. A close neighbor of an infeasible solution may be a good feasible solution.

In our computation, w is randomly given as $w \in [1.01, 1.20]$ for infeasible schedules. And then another argument is "why not a large additive penalty is used to drive the search to feasible solutions?" In our computational tests, we have tried to use large penalty values, but which only increases the search time without efficient improvement to the solution quality. Near an infeasible solution there may be good feasible solutions. Large penalty values adopted drive the search quickly away from the region around the infeasible solution.

Comparison of OGA and IGA for Examples 1–3

In this sub-section, first, IGA is applied to Examples 1 and 2 for minimizing *PT*. In Examples 1 and 2, all orders can be completed before their due dates, so that only one IGA procedure to minimize *PT* is required. In the IGA procedure, every chromosome is synthesized into a schedule according to the following requirements for minimization of

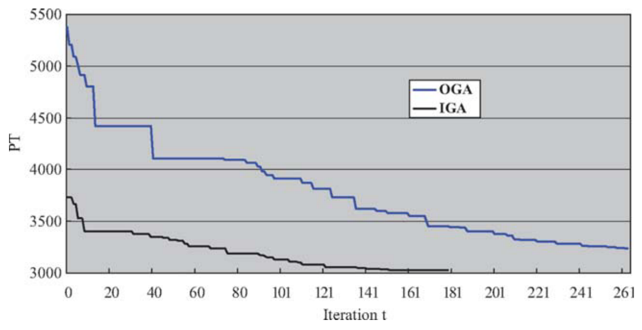


Figure 4. Convergence process of OGA and IGA for Example 1 (min. *PT*, *N* = 24).

[Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

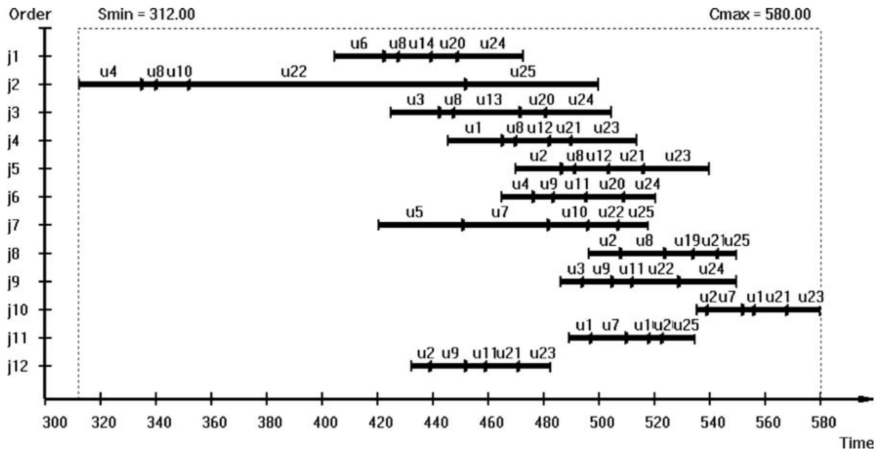


Figure 5. A schedule for the 12-order/25-unit instance in Example 1 by IGA (min. $PT = 1031.90$).

PT: (1) backward assignment;[25] (2) technique (similar to active scheduling) to reduce the idle times of the units and intermediate storage time of the semi-finished products[25]; (3) for each stage, choose a proper position to assign an order according to Rule 8. The parameters used in IGA are $C_r = 0.7$, $M_r = 0.3$. The termination condition for IGA is that the algorithm stops when the objective value difference between the worst chromosome and the best one in the current generation is equal to or less than small value, like 0.001.

Next, IGA is applied to Examples 1 and 2 for minimizing the total flow time *F*. In the IGA procedure, every chromosome is synthesized into a schedule according to the requirements for minimization of the total flow time: the forward assignment strategy, the active scheduling technique and the position selection rule – Rule 7. The parameters and the termination condition are the same as previous.

And then, IGA is applied to Example 3 for minimizing *PT*. Example 3 is an example with tardy orders at the optimum, hence first minimize *T* and set soft due dates of the orders, then minimize *PT*.

Table 2 presented the best results of Example 1 (varying problem size from 5 to 24 orders) solved by IGA, OGA and MILP. Table 2 shows that both IGA and OGA perform much better than MILP in terms of solution quality and search time. Compared to OGA, IGA increases solution quality, though it requires a little more search time due to the position selection. With the increasing of the problem size, the difference between IGA and OGA increases. For the large-size 24-order instance, IGA performs much better than OGA, see Figure 4, which shows that from the very beginning of the search process, IGA obtain much better solution than OGA. Figure 5 shows a schedule obtained by IGA, where no idle times are seen. The performance stability of IGA can be seen later from the average results of a number of computational tests in Tables 3 and 4.

To test the performance stability of IGA, 10 computational tests have been conducted for each problem size in Examples 1 and 2. Mean deviations from the best and mean computational (CPU) times are regarded as average performance criteria of the algorithm.

Table 3 shows the average performance of IGA and OGA for Examples 1 and 2 under different problem sizes in minimizing *PT*. For Example 1, it can be noted that with the increasing of the problem size, the corresponding mean deviations from best by IGA increase (<3% for all instances), far lower than those of OGA (up to 15.56% for the 24-order

Table 3. Average Performance of IGA and OGA for Examples 1 and 2 (min. *PT*)

			Best *PT*			Mean *dev. from best*		Mean CPU Time (s)	
	$N \times M$	*popsize*	IGA	OGA	*Diff2*	IGA	OGA	IGA	OGA
Example 1	5×5	200	499.40	499.40	0.00	0.00	0.00	0.05	<1.0
	8×5	200	730.90	732.90	0.27	0.12	0.31	0.71	2.0
	10×5	300	863.90	876.40	1.45	0.15	1.88	1.56	3.7
	12×5	400/300	1031.90	1038.90	0.68	0.62	1.59	4.66	4.6
	16×5	600	1763.80	1788.80	1.42	0.84	2.73	13.71	17.5
	20×5	600	2413.50	2456.80	1.79	1.02	6.63	34.03	28.2
	24×5	600	3024.50	3239.30	7.10	2.42	15.56	58.55	36.1
Example 2	5×5	200	633.90	633.90	0.00	0.00	0.00	0.12	<1
	8×5	200	951.90	951.90	0.00	0.16	0.16	0.76	1.45
	10×5	300	1158.90	1172.50	1.17	0.35	2.22	2.52	4
	12×5	400/300	1429.50	1436.90	0.52	0.84	4.32	6.10	6.3
	16×5	600	2352.30	2444.80	3.93	0.96	8.40	19.71	18.2
	20×5	600	3224.60	3544.20	9.91	1.32	15.64	38.27	30.7
	24×5	600	4108.60	4797.70	16.77	2.94	24.58	49.31	31.3

Table 4. Average Performance of IGA and OGA for Examples 1 and 2 (min. F)

			Best F			Mean *dev. from best*		Mean CPU Time (s)	
	$N \times M$	*popsize*	IGA	OGA	*Diff2*	IGA	OGA	IGA	OGA
Example 1	5 × 5	200	529.90	529.90	0.00	0.00	0.00	0.19	<1.00
	8 × 5	200	815.80	815.80	0.00	0.51	0.82	1.02	<1.00
	10 × 5	300	1007.00	1012.60	0.55	0.95	2.15	2.82	2.2
	12 × 5	400/300	1239.30	1239.30	0.00	1.60	2.79	8.17	5.9
	16 × 5	600	2052.60	2064.00	0.55	2.28	5.00	30.17	22.2
	20 × 5	600	2750.60	2855.00	3.80	3.53	8.43	65.35	45.9
	24 × 5	600	3785.80	3832.70	1.24	4.60	7.28	67.93	57.6
Example 2	5 × 5	200	684.30	691.70	1.08	0.00	1.08	0.20	<1.00
	8 × 5	200	1091.30	1098.60	0.67	0.21	2.13	0.88	2.1
	10 × 5	300	1364.00	1425.20	4.49	0.68	7.31	3.38	5.4
	12 × 5	400/300	1696.30	1823.90	7.52	1.37	10.16	8.04	8.8
	16 × 5	600	2715.60	3000.70	10.50	2.04	14.82	36.26	23.9
	20 × 5	600	3744.60	4477.80	19.58	3.20	25.31	48.48	25.4
	24 × 5	600	5240.70	5864.30	11.90	3.44	18.74	55.40	27.3

instance). For OGA, with the increasing of problem size, the mean deviations from best increase quickly. For Example 2, the mean deviations from best by IGA are still <3% for all instances, but those by OGA range from 0.00% to 24.58%. Table 4 shows the average performance of IGA and OGA for Examples 1 and 2 under different problem sizes in minimizing F. From Tables 3 and 4, the following observations can be made:

(1) Through comparing the best solutions for Examples 1 and 2, IGA exhibits its dominance over OGA more evidently in Example 2. The reason is that Example 1 is relatively simple, not involving different unit release times, unit setup times, and order release times.

(2) Through comparing mean deviations from best, IGA shows more performance stability in solving large-size instances than OGA. This owes to the new position selection rules used. The difference between IGA and OGA lies in the decoding procedure. Other operators are the same, and the parameter settings are almost the same.

(3) It takes a little more CPU time for IGA to solve the problems. The position selection needs CPU time.

Because the lower mean deviations from best of IGA indicate that IGA has more stable search performance than OGA, IGA is subsequently used for solving much larger general MMSP, e.g., a 50-order problem (Example 6).

Table 5. Solutions to the 5-Order Instance in Example 3 (First min. T, then min. PT)

		A Schedule (min. T = 179.6)				An Infeasible Schedule (min. PT = 602.9)				A Feasible Schedule (min. PT = 602.9)			
Order	d_j	Unit	Start Time	End Time	T_j	Unit	Start Time	End Time	T_j	Unit	Start Time	End Time	T_j
j1		u3	5.0	31.1		u2	−**40.1**	−14.0		u2	5.0	31.1	
j1		u8	31.1	37.1		u9	−14.0	−8.0		u8	31.1	37.1	
j1		u13	37.1	51.6		u13	−8.0	6.5		u10	37.1	51.6	
j1		u21	51.6	67.1		u20	6.50	22.0		u21	51.6	67.1	
j1	50	u23	67.1	**95.1**	**45.1**	u24	22.0	50.0	0.0	u24	67.1	95.1	0.0
j2		u1	6.0	37.0		u5	−**128.5**	−97.5		u1	6.0	37.0	
j2		u9	37.0	43.0		u8	−97.5	−91.5		u9	37.0	43.0	
j2		u12	43.0	57.5		u10	−91.5	−77.0		u12	43.0	57.5	
j2		u22	57.5	181.5		u22	−77.0	47.0		u22	57.5	181.5	
j2	100	u25	181.5	**234.5**	**134.5**	u25	47.0	100.0	0.0	u25	181.5	234.5	0.0
j3		u5	2.0	28.1		u4	98.1	124.2		u1	98.1	124.2	
j3		u7	28.1	50.1		u9	124.2	130.2		u9	124.2	130.2	
j3		u11	50.1	76.6		u12	130.2	156.7		u11	130.2	156.7	
j3		u20	85.5	100.8		u20	156.7	172.0		u20	156.7	172.0	
j3	200	u23	108.8	136.8	0.00	u23	172.0	200.0	0.0	u24	172.0	200.0	0.0
j4		u2	2.0	30.0		u1	209.6	237.6		u2	209.6	237.6	
j4		u8	46.1	52.1		u8	237.6	243.6		u8	237.6	243.6	
j4		u10	52.1	66.6		u14	243.6	258.1		u11	243.6	258.1	
j4		u20	66.6	80.5		u21	258.1	272.0		u20	258.1	272.0	
j4	300	u24	134.0	162.0	0.00	u23	272.0	300.0	0.0	u23	272.0	300.0	0.0
j5		u4	6.0	31.0		u3	308.0	333.0		u1	308.0	333.0	
j5		u9	51.0	57.0		u9	333.0	339.0		u8	333.0	339.0	
j5		u14	57.0	71.5		u11	339.0	353.5		u10	339.0	353.5	
j5		u21	81.5	100.0		u21	353.5	372.0		u20	353.5	372.0	
j5	400	u24	100.0	128.0	0.00	u23	372.0	400.0	0.0	u23	372.0	400.0	0.0
			(d_1 = 50, d_2 = 100)				(d_1 = 50, d_2 = 100)				(d'_1 = 95.10, d'_2 = 235.50)		

Table 6. Best Results of Example 3 by IGA and OGA (min. *PT*)

$N \times M$	*popsize*	Part One: min. *T* (IGA) CPU (s)	Part One: min. *T* (IGA) *T*	Part Two: min. *PT* (IGA) CPU (s)	Part Two: min. *PT* (IGA) Best *PT*	Part Two: min. *PT* (OGA) CPU (s)	Part Two: min. *PT* (OGA) Best *PT*	*Diff2*
5 × 5	200	0.055	179.60	<0.055	602.90	<1	602.90	0.00
8×5	200	0.22	179.60	0.38	874.90	<1	874.90	0.00
10×5	300	0.50	179.60	1.48	1036.90	2	1039.90	0.29
12×5	400	1.05	179.60	3.52	1233.40	2	1248.90	1.26
16×5	600	2.97	179.60	14.01	1860.80	18	1889.30	1.53
20×5	600	5.06	179.60	35.38	2536.80	28	2806.40	10.63
24×5	600	7.47	179.60	58.24	**3328.70**	32	3845.10	15.51

j_1 and j_2 are two tardy orders at optimum (min. *T*), let $d'_1 = 95.10$, $d'_2 = 235.50$ when min. *PT*

As seen in He and Hui,[25] through minimizing *T* for Example 3, it is found to be an example with tardy orders at the optimum. When using IGA to minimize *T* of Example 3 with various problem sizes, an identical minimum *T* = 179.60 is obtained. Orders j1 and j2 are two tardy orders at the optimum, $C_1 = 95.10$ ($>d_1 = 50$), $C_2 = 234.50$ ($>d_2 = 100$). When we keep the due dates of j1 and j2 unchanged, still to be 50 and 100, and minimize *PT*, we get a schedule with negative start times of orders j1 and j2, an infeasible schedule shown in the middle of Table 5. However, when we adopt soft due dates, let $d'_1 = 95.10$, $d'_2 = 234.50$, and minimize *PT* using Eq. 6, a feasible schedule is obtained.

Table 6 shows the results of different size instances in Example 3. For all the instances, we first minimize *T* and set the soft due dates, and then based on the soft due dates minimize *PT* using Eq. 6. We have noticed the following facts:

(1) All the instances have the same optimal tardiness *T* = 179.60, and it is easy to get the tardiness objective (with short search times).

(2) When minimizing *PT*, the search times are longer than those in minimizing *T*; moreover, with problem size increasing, infeasible schedules with negative start times are found sometimes.

We still conducted 10 computational tests for each problem size in Example 3 when minimizing *PT*. In one test for the 8-order/25-unit instance, two tests for the 10-order/25-unit instance, three tests for the 16-order/25-unit instance, three tests for the 20-order/25-unit instance and 8 tests for the 24-order/25-unit instance (see Table 7), the final solutions were infeasible schedules with negative start times. That is why we have not given the deviations of the results in Table 6.

When solving the large-size instances, in case of infeasible schedules with negative start times, the penalty method with Eq. 7 can be used. Table 7 shows the performance contrast of two methods in solving the 24-order instance. With the use of the penalty method, the average search time becomes longer, but in every test, feasible solutions are achieved at last.

A Global Search Framework

Besides the improvement of the single GA, an iterative framework is developed to achieve the good feasible solutions of large-scale MMSP with tight due dates. In this framework, both evolved solutions and dynamic statistic information (the so-called evolutionary gradient) are harnessed to guide the global search.

As well known, single meta-heuristic is problem dependent, its robustness is limited, and premature convergence is another main drawback, especially in case for large-scale problems. Hybridization, parallelization, and other search frameworks are the main effective measures to enhance the robustness and to overcome the drawback of premature convergence. The proposed framework aims to address two issues: the first goal is to solve larger problems within reasonable computational time; the second is to make the algorithm more robust and less calibration efforts. In fact, this

Table 7. Contrast of Two Methods in Solving the 24-Order Instance in Example 3

Test	min. *PT* Using Eq. 6 *t*	min. *PT* Using Eq. 6 CPU (s)	min. *PT* Using Eq. 6 *PT*	min. *PT* Using Eq. 6 INF or F	min. *PT* Using Eq. 7 *t*	min. *PT* Using Eq. 7 CPU (s)	min. *PT* Using Eq. 7 *PT*	min. *PT* Using Eq. 7 INF or F
1	167	67.00	3364.70	INF	291	94.00	3329.70	F
2	133	47.64	3440.40	INF	254	84.00	3431.10	F
3	147	47.36	3373.60	INF	**131**	**43.68**	**3317.00**	**F**
4	108	35.05	3440.30	INF	168	56.20	3372.60	F
5	132	42.53	3424.80	INF	155	51.70	3332.40	F
6	179	58.24	**3328.70**	**F**	152	50.38	3388.80	F
7	177	60.00	3349.90	INF	178	60.00	3344.00	F
8	188	60.00	3390.70	INF	128	42.53	3378.80	F
9	108	35.00	3469.20	INF	177	57.80	3409.60	F
10	146	46.98	3409.80	F	182	61.00	3385.60	F
Mean	148.5	49.98	3399.21		181.6	60.13	3368.96	

INF, infeasible; F, feasible.

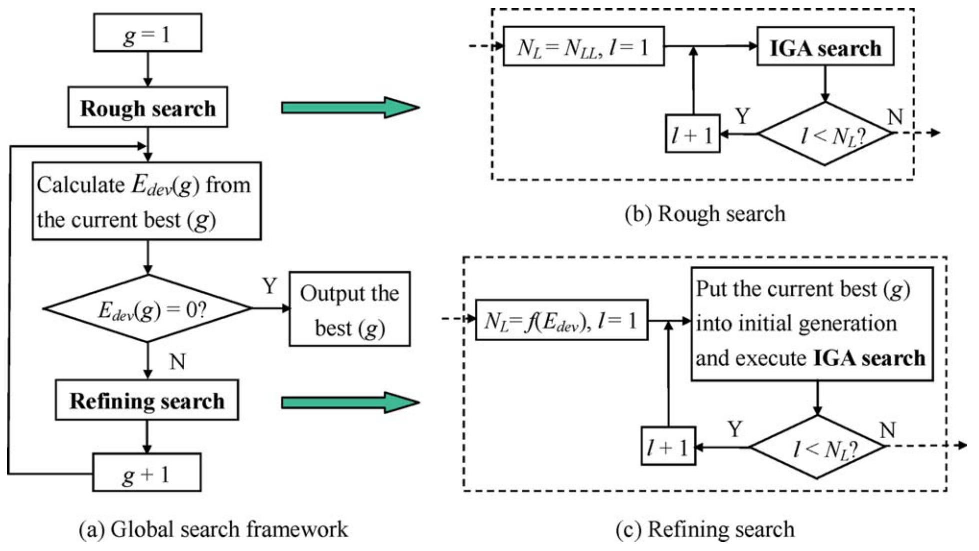


Figure 6. Flow chart of the global search framework.

[Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

framework has realized some parallel ideas in the sequential environment.

With the consideration of practical application, the user is more willing to run the scheduling program for only one time and then obtain the expected scheduling solution, other than running the program for a number of times and then selecting the best solution. However, on the other hand, the distinction between the solutions from a number of computational tests is the impetus for us to pursuit much better solutions. In other words, the existing distinction implies that much better solutions are possible if more computational tests are conducted. On the contrary, if solutions with the same objective value are finally achieved in all computational tests, this may implies that the algorithm is limited to the current conditions with the restricted ability just to find the current solutions.

At the moment of seeing the distinction between the solutions, what we should consider is how to effectively conduct the subsequent computational tests and when to stop computing. Regarding these two questions, we propose the concept of evolutionary gradient and then devise a GSF. Under such a solution strategy, when the evolutionary gradient vanishes, the algorithm stops proceeding ahead.

Evolutionary gradient

As well known, the concept of gradient is adopted in numerical methods, where the gradient is regarded as the criterion to determine the termination of the algorithm—if the gradient turns to be zero, the optimum or local optimum is deemed to have been found. Similarly, in evolutionary algorithms, the distinction between the solutions obtained so far from computational tests can be treated as the criterion to determine the termination of the algorithms, thus the concept of evolutionary gradient is proposed.

When a number of computational tests are carried out to evaluate the algorithm, the RD of each test can be calculated with respect to the current best solution:

$$\text{RD}\,(\%) = 100 \times \frac{\text{objective value} - \text{current best}}{\text{current best}}. \quad (8)$$

Table 8. Results of Examples 1, 2, and 3 by GSF (min. *PT*)

		Rough Search			Final Refining Search			
	$N \times M$	N_L	CPU (s)	Current Best *PT*	Final Best *PT*	CPU (s)	g	N_L
Example 1	16 × 5	20	218	1768.70	1759.80	506	3	20
	20 × 5	20	456	2415.40	2412.50	870	3	20
	24 × 5	20	807	3020.10	2982.10	3079	8	20
Example 2	16 × 5	20	314	2351.30	2345.30	645	3	20
	20 × 5	20	500	3183.30	3170.30	1389	5	20
	24 × 5	20	807	4134.80	4063.30	3459	8	20
Example 3*	16 × 5	20	254	1843.30	1834.30	814	4	20
	20 × 5	20	442	2521.80	2517.80	848	3	20
	24 × 5	20	827	3297.70	3284.00	2900	6	20

*min. *PT* with soft due dates.

Table 9. Results of Example 2 by GSF for (min. F)

	Rough Search			Final Refining Search			
$N \times M$	N_L	CPU Time (s)	Current Best F	Final Best F	CPU Time (s)	g	N_L
16×5	20	444	2783.70	2694.90	1507	7	20
20×5	20	550	3914.90	3683.30	3328	11	20
24×5	20	844	5166.90	4885.50	6575	14	20

Let N_L denote the number of computational tests of the single IGA. The mean RD is defined as evolutionary gradient, denoted by E_{dev}:

$$E_{dev} = \frac{1}{N_L}\sum_{l=1}^{N_L} \mathrm{RD}_l, \tag{9}$$

where RD_l is the RD of test l calculated by Eq. 8. In fact, E_{dev} can be regarded as an important sign which indicates whether it is necessary to conduct more computational tests for better solutions. If the N_L tests obtain solutions with the same objective value, then $E_{dev} = 0$, which implies that it is not necessary to carry out more tests; else, $E_{dev} > 0$, which indicates the probability of better solutions through another number of computational tests. The initial N_L depends on the problem size. For large-size problems, we usually witness $E_{dev} > 0$, and an additional number of runs should find better solutions. The additional N_L can be determined by the subsequent Eq. 10.

Global search framework

Based on the concept of evolutionary gradient, two phases of computational tests are organized. In phase 1, called rough search, N_L tests of IGA are carried out, and evolutionary gradient E_{dev} is calculated. And then go on with phase 2 consisting of one or more refining searches, each refining search involves $N_L = f(E_{dev})$ tests of IGA, until the evolutionary gradient turns to be zero ($E_{dev} = 0$). The rough search and refining searches can be considered as local searches, while the whole organization of the two phases is called the GSF. Each rough search or refining search is seen as a global iteration, denoted by g; and a single IGA test as a local iteration, denoted by l. (t denotes the iterations in one IGA search).

The procedure of the GSF is shown in Figure 6(a). The rough search, as shown in Figure 6b, comprises N_L tests of IGA, each of which starts with an initial generation including *popsize* random chromosomes, Each refining search, as shown in Figure 6c, comprises $N_L = f(E_{dev})$ tests of IGA, each test starts with an initial generation consisting of the current best chromosome from last global iteration g and (*popsize*-1) random chromosomes. $f(E_{dev})$ can be determined by:

$$f(E_{dev}) = \begin{cases} N_{\mathrm{LL}}\left(1+\frac{1}{E_{dev}}\right), & \text{if } E_{dev} \geq \frac{N_{\mathrm{LL}}}{N_{\mathrm{LU}}-N_{\mathrm{LL}}} \\ N_{\mathrm{LU}}. & \text{else} \end{cases}, \tag{10}$$

where N_{LL} and N_{LU} are respectively the lower and upper bounds of the number of IGA tests. N_L increases with the decreasing of E_{dev}, but has an upper bound. N_L may also be set as a constant parameter in the search process. In this work, N_L is given as a constant number.

In addition to IGA, other hybrid solution methods can be also incorporated into the GSF. Using such a framework, better solutions can be obtained in succession, until the evolutionary gradient vanishes. One may argue: "What is the guarantee that the framework would converge in finite time, what if the E_{dev} never vanishes?" The evolutionary mechanism of GA guarantees the convergence of the framework, which is not the content of this paper. We use the varying E_{dev} to represent the convergent trend. In all our computational tests, the case that the E_{dev} never vanishes has never been witnessed. The optimality of this framework is associated with the search technique(s) incorporated. IGA is a type of meta-heuristics which do not guarantee absolute optimality, so the GSF does not guarantee absolute optimality either. With the evolved solutions and evolutionary gradient to guide and control the search process, the framework can guarantee convergence speed and avoid repetitious global iterations that do not improve the solutions.

Results of Examples 1–3 by GSF

Table 8 presents the results of Examples 1–3 solved by GSF (min. PT), Table 9 provides the results of Example 2 solved by GSF (min. F). From these three tables, it can be noted that, through the search of GSF, all of the instances gain better solutions. For the large-size instances, GSF has improved the solution quality a lot.

Examples with Tight Due Dates

To test the performance of the IGA, it has been applied to solving much larger general MMSP, Examples 4–6. Tables B-1 and B-2 in Appendix B (see Supporting Information)

Table 10. Results of Examples 4, 5, and 6 by IGA

		min. PT			min. F		
Example	*popsize/xsize/msize*	Best PT	Mean *dev. from best*	Mean CPU (s)	Best F	Mean *dev. from best*	Mean CPU (s)
4	550/400/150	2874.46	3.76	101.50	3577.39	3.56	77.93
5	500/400/100	4127.3	4.16	78.58	4829.63	3.96	75.33
6	550/400/150	5331.57	4.59	119.90	6426.72	4.47	84.60

Table 11. Old Due Dates and New Due Dates for Examples 4′, 5′, and 6′

Order	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Old d_j	215	248	209	230	217	220	211	243	211	225	249	231	244	227	242	205	203
New d_j	154	166	189	194	165	197	169	158	187	174	153	174	167	173	159	198	197
Order	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
Old d_j	212	231	222	226	210	202	247	245	226	244	202	214	248	221	202	242	240
New d_j	196	178	151	165	155	178	164	177	193	153	179	151	197	152	180	153	173
Order	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	
Old d_j	221	201	226	219	210	222	214	218	225	226	245	247	207	246	232	204	
New d_j	165	173	160	185	195	199	182	173	195	189	152	161	159	151	161	170	

present the problem data. There are 21 units available to process up to 50 customer orders. Each order needs to go through up to five processing stages. Units 1–4 are the parallel processing units in Stage 1, Units 5–9 in Stage 2, Units 10–12 in Stage 3, Units 13–18 in Stage 4, and Units 19–21 in Stage 5. So the numbers of units in the five stages are {4, 5, 3, 6, 3}. In Table B-1 and Table B-2, the problem data p_{ju}, ut_u, ur_u, c_{ij}, d_j, and or_j are generated randomly, $p_{ju} \in (5.00, 20.00)$, $ut_u \in [0, 8)$, $ur_u \in [0, 4)$, $c_{ij} \in (0.10, 2.00)$, $d_j \in [200, 250)$, $or_j \in [0, 6)$, without CP constraints. In Examples 4–6, the number of customer orders and the number of processing stages are different:

- Example 4: the former 30 orders are to be processed through all the five stages.
- Example 5: the former 40 orders are to be processed through the former four stages.
- Example 6: all the 50 orders are to be processed through the former three stages.

Solution for Examples 4–6

With the original due dates in Appendix B, zero total tardiness is easy to be obtained in Examples 4–6, which means that no tardy orders exist in the optimized schedules. Therefore, minimizations of *PT* for these examples can be performed directly without setting soft due dates. Table 10 summarizes the results of these three examples by IGA for minimization of *PT* and minimization of the total flow time *F* (10 tests conducted for each example). Observation on the mean deviations from best in Table 10 shows that they are within a lower range (<5%). That is to say, IGA still demonstrates stable search ability to solve these much larger MMSP.

IGA for examples with tight due dates

To test the performance the IGA in solving problems with tardy orders at the optimum, the old due dates in Appendix B are changed into the new due dates shown in Table 11. The new due dates are generated randomly, $d_j \in [150, 200)$. With these new due dates, the Examples 4–6 are changed into Examples 4′, 5′, and 6′ correspondingly. Except the due dates, other problem data are not changed.

With the new due dates, through minimizing the total tardiness of Examples 4′, 5′, and 6′ by IGA (see Table 12), zero total tardiness can be obtained for Example 4′. Therefore, Example 4′ does not need to set the soft due dates in minimizing *PT*. However, for Examples 5′ and 6′, certain total tardiness exists ($T > 0$). After the minimization of tardiness *T*, the soft due dates are set as shown in Table 13. These are the cases with tight due dates. For such cases, if the total tardiness and the earliness-related objective are considered simultaneously in scheduling, it is a challenge to the solution method. By using the proposed procedure—first minimize *T* (Step 1) and set soft due dates, next minimize *PT* (Step 2), if the solution method is effective and efficient enough, both steps can solve the problem well. Otherwise, at Step 2, feasible solutions can not be obtained.

When minimizing *PT* of Example 4′ using Eq. 5, the situation is similar to Example 3: among 10 tests of computation, four tests find feasible solutions, but other six tests get infeasible solutions. However, when minimizing *PT* of Examples 5′ and 6′ with the soft due dates, the situation has changed—it is difficult for IGA to find feasible solutions, even increasing the population size (*popsize*) and the number of computational tests. The following scenarios have been tested to get feasible solutions:

(1) Scenario 1(rough search): perform 10 IGA tests minimizing *PT* using Eq. 6.

(2) Scenario 2: perform 10 IGA tests minimizing *PT* using Eq. 7.

(3) Scenario 3: first (rough search), perform 10 computational tests minimizing *PT* using Eq. 6, second (refining search), put the best solution (the corresponding chromosome) of the previous 10 tests into the initial generations of coming computational tests as an excellent individual, and then perform 10 computational tests minimizing *PT* using Eq. 7. If necessary, the new best solution can be put into the initial generations of new coming computational tests, and conduct another one or more refining searches, until feasible

Table 12. Results of Examples 4′, 5′, and 5′ by IGA (min. *T*)

Example	*popsize*/*xsize*/*msize*	Best *T*	Mean *T*	Mean CPU time (s)	Tests
Example 4′	550/400/150	0.00	21.47	76.10	10
Example 5′	500/400/100	120.88	168.09	78.95	10
Example 6′	550/400/150	398.77	509.40	79.90	10

Table 13. Tardy Orders and Soft Due Dates in Examples 5′ and 6′ (min. T by IGA)

	Order	d_j	C_j	T_j	d'_j (=C_j)
Example 5′ (T = 120.88)	j2	166.00	202.62	36.62	202.62
	j9	187.00	189.41	2.41	189.41
	J12	174.00	174.37	0.37	174.37
	j14	173.00	177.39	4.39	177.39
	j16	198.00	198.59	0.59	198.59
	j18	196.00	219.50	23.50	219.50
	j19	178.00	178.47	0.47	178.47
	j25	177.00	197.20	20.20	197.20
	j28	179.00	211.33	32.33	211.33
Example 6′ (T = 398.77)	j2	166.00	238.11	72.11	238.11
	j5	165.00	218.52	53.52	218.52
	j6	197.00	201.61	4.61	201.61
	j12	174.00	175.59	1.59	175.59
	j14	173.00	192.53	19.53	192.53
	j25	177.00	247.00	70.00	247.00
	j26	193.00	205.86	12.86	205.86
	j27	153.00	219.71	66.71	219.71
	j28	179.00	230.83	51.83	230.83
	j36	173.00	180.77	7.77	180.77
	j40	199.00	210.57	11.57	210.57
	j43	195.00	203.72	8.72	203.72
	j46	161.00	178.95	17.95	178.95

Table 14. Different Scenarios for the Minimization of PT of Examples 4′, 5′, and 6′

	Example 4′		Example 5′		Example 6′	
Scenario	Best PT	INF or F	Best PT	INF or F	Best PT	INF or F
Scenario 1	2865.32	F	3713.46	INF	3810.23	INF
Scenario 2	3044.06	F	N/A	N/A	N/A	N/A
Scenario 3						
Rough	2865.32	F	3713.46	INF	3810.23	INF
Refining	2865.32	F	3873.70	F	3792.54	F

Table 15. Results of Examples 5′ and 6′ by GSF (min. T with New Due Dates)

	Rough Search			Final Refining Search			
Example	N_L	CPU (s)	Current Best T	Final Best T	CPU (s)	g	N_L
Example 5′	20	1310	112.49	**46.71**	6482	10	20
Example 6′	30	1927	376.66	**252.18**	13434	17	30

Table 16. Tardy Orders and Soft Due Dates in Examples 5′ and 6′ (min. T by GSF)

	Order	d_j	C_j	T_j	d'_j (=C_j)
Example 5′ (T = 46.71)	j2	166.00	169.17	3.17	169.17
	j4	194.00	195.01	1.01	195.01
	j6	197.00	198.78	1.78	198.78
	j16	198.00	206.61	8.61	206.61
	j18	196.00	211.30	15.30	211.30
	j19	178.00	191.05	13.05	191.05
	j34	173.00	176.79	3.79	176.79
Example 6′ (T = 252.18)	j2	166.00	239.06	73.06	239.06
	j4	194.00	218.85	24.85	218.85
	j16	198.00	201.74	3.74	201.74
	j18	196.00	222.27	26.27	222.27
	j19	178.00	232.48	54.48	232.48
	j25	177.00	216.59	39.59	216.59
	j28	179.00	206.44	27.44	206.44
	j30	197.00	198.60	1.60	198.60
	j34	173.00	174.15	1.15	174.15

Table 17. Results of Examples 4′, 5′, and 6′ by GSF (min. *PT* with Soft Due Dates)

	Rough Search			Final Refining Search			
	N_L	CPU (s)	Current best *PT*	Final best *PT*	CPU (s)	*g*	N_L
Example 4′	20	1441	2872.97 (F)	2800.18 (F)	6498	9	20
Example 5′	30	2460	3674.54 (INF)	3503.63 (F)	20476	13	30
Example 6′	**30**	2772	4304.02 (INF)	**3971.46 (INF)**	20630	18	30
	50	4061	4136.14 (INF)	**3898.51 (F)**	32460	19	50

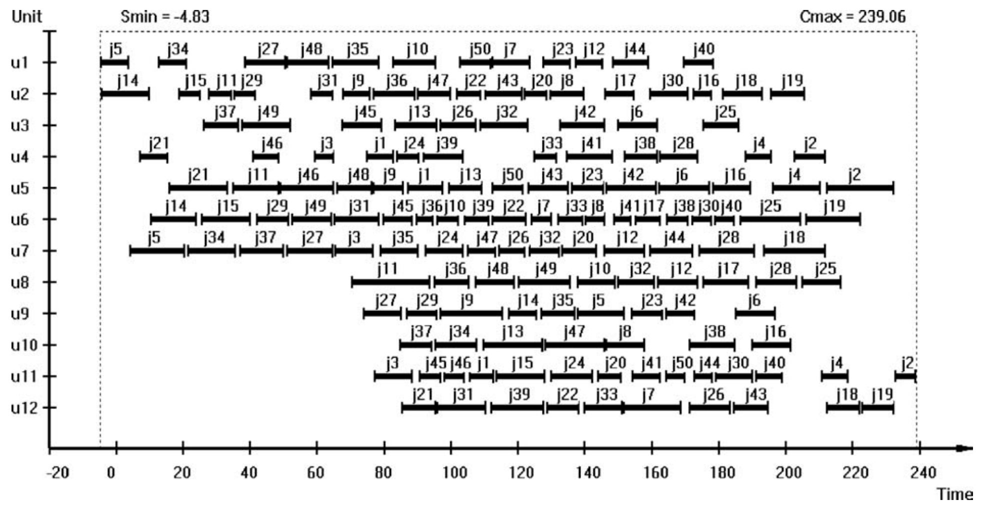


Figure 7. An infeasible schedule for Example 6′ by GSF (min. *PT* = 3971.46).

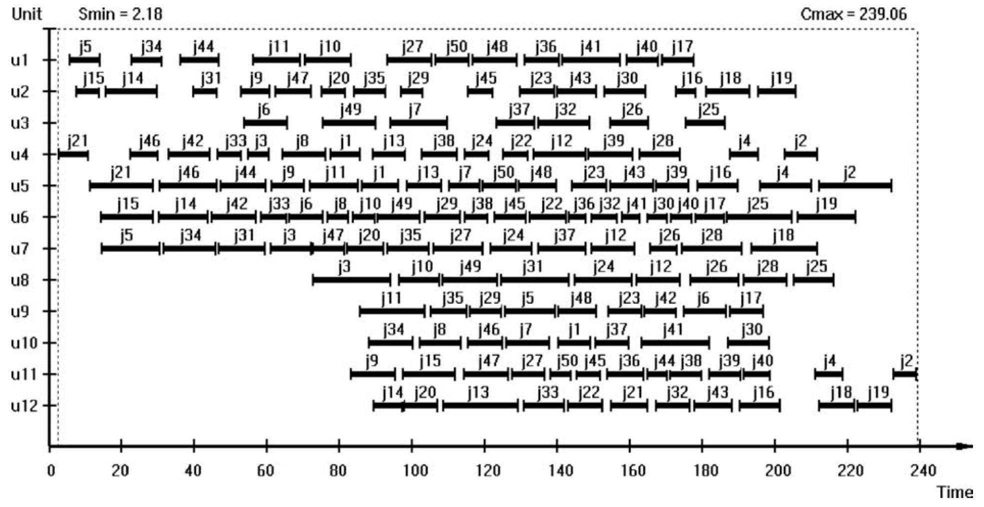


Figure 8. A feasible schedule for Example 6′ by GSF (min. *PT* = 3898.51).

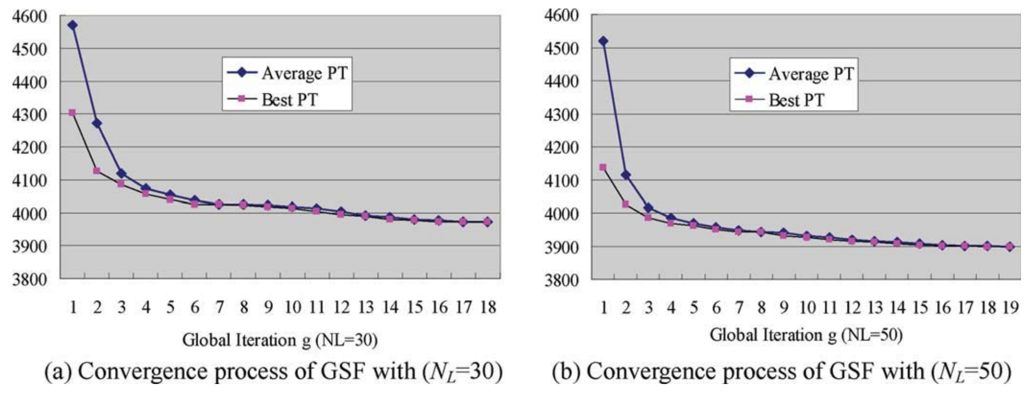


Figure 9. Comparison of the convergences of GSF with N_L = 30 and N_L = 50 for Example 6′.

[Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

Table 18. Results in the Rough Search and Refining Search 1 of GSF with $N_L = 30$ for Example 6′

Summary of the Results in Rough Search ($g = 1$)					Summary of the Results in Refining Search 1 ($g = 2$)				
Test *l*	*T*	CPU (s)	*PT*	RD_l	Test *l*	*t*	CPU (s)	*PT*	RD_l
1	97	65.00	4717.64	9.61	1	61	54.73	4286.88	3.90
2	125	85.00	4501.17	4.58	2	54	46.04	4289.41	3.96
3	133	96.00	4627.22	7.51	3	62	52.58	4284.16	3.83
…	…	…	…	…	…	…	…	…	…
12	176	117.00	**4304.02**	**0.00**	19	73	43.63	**4126.16**	**0.00**
…	…	…	…	…	…	…	…	…	…
29	117	95.00	4443.12	3.23	29	70	42.75	4292.01	4.02
30	137	136.00	4414.57	2.57	30	105	62.00	4247.71	2.95
Average	125.97	92.41	4570.48	6.19	Average	70.27	49.59	4270.75	3.50

Table 19. Comparison of the Search Processes of GSF with $N_L = 30$ and $N_L = 50$ for Example 6′

$N_L = 30$						$N_L = 50$					
g	Av *t*	Av CPU	Av *PT*	E_{dev}	Best *PT*	*g*	Av *t*	Av CPU	Av *PT*	E_{dev}	Best *PT*
1	125.97	92.41	4570.48	6.19	4304.02	1	134.12	81.23	4519.38	9.27	4136.14
2	70.27	49.59	4270.75	3.50	4126.16	2	72.68	43.87	4115.27	2.20	4026.80
3	57.37	34.51	4117.52	0.81	4084.53	3	58.60	36.49	4015.57	0.72	3986.77
4	61.60	37.08	4072.47	0.38	4056.96	4	51.46	31.27	3985.38	0.39	3969.98
5	54.80	33.17	4052.79	0.36	4038.47	5	52.80	32.00	3969.47	0.21	3961.34
6	59.03	35.76	4037.24	0.28	4026.18	6	57.58	34.68	3958.81	0.22	3950.29
7	59.83	36.29	4025.98	0.01	4025.58	7	50.54	30.64	3949.31	0.12	3944.55
8	61.73	37.43	4025.17	0.07	4022.53	8	50.96	39.17	3944.50	0.04	3943.01
9	56.20	34.15	4022.26	0.09	4018.43	9	55.18	33.24	3940.23	0.19	3932.55
10	58.00	35.15	4017.87	0.12	4013.17	10	50.38	30.48	3931.98	0.12	3927.03
11	51.97	31.65	4012.48	0.21	4003.82	11	48.70	29.58	3926.69	0.14	3921.13
12	54.27	32.89	4003.15	0.24	3993.52	12	45.80	27.86	3920.97	0.15	3915.39
13	56.00	33.89	3992.81	0.12	3988.20	13	45.00	27.59	3915.04	0.06	3912.63
14	50.70	30.72	3986.99	0.19	3979.36	14	46.76	28.26	3911.92	0.11	3907.72
15	55.60	33.72	3979.13	0.06	3976.53	15	48.72	29.33	3907.37	0.08	3904.19
16	55.63	33.66	3976.40	0.10	3972.67	16	49.82	30.09	3903.76	0.06	3901.31
17	57.63	34.80	3971.66	0.01	3971.46	17	45.76	27.69	3901.29	0.01	3901.03
18	50.87	30.83	3971.46	0.00	3971.46	18	46.38	28.02	3900.97	0.06	3898.51
						19	45.34	27.69	3898.51	0.00	3898.51

(and good enough) solutions are found, or until no improvement on solution quality.

Table 14 presents the results of the three examples solved with different scenarios. From Table 14, it is observed that: (1) Example 4′ is solved to feasibility by all the scenarios; (2) Examples 5′and 6′ are not solved to feasibility by Scenarios 1 and 2, but through the refining searches in Scenario 3, feasible and further better solutions are obtained; (3) in solving Example 5′ with Scenario 3, the objective value of the feasible solution obtained in the refining search is worse than that of the infeasible solution obtained in the rough search.

In fact, it is Scenario 3 applied to Examples 4′, 5′, and 6′ that inspires the idea of GSF. A large MMSP with tight due dates is difficult to solve to feasibility (let alone optimality) by just using a single genetic search like Scenario 1 or Scenario 2. The main cause for this difficulty is the problem of premature convergence existing in evolutionary algorithms. One may argue that adjusting the crossover rate and mutation rate may enhance the search ability of the algorithms. From our experience, the effect of tuning parameters of the evolutionary algorithms is limited. But harnessing the evolved solutions to guide restarted searches is indeed a viable idea, as seen in Scenario 3.

GSF for Examples with tight due dates

As stated above, under the soft due dates, there must be feasible solutions when minimizing the earliness related objectives with the backward assignment strategy. To verify this statement, in this sub-section, GSF is applied to the examples with tight due dates. First, GSF is used to minimize *T* for Examples 5′ and 6′, and the soft due dates are set consequently. And then GSF is used to minimize *PT* for Examples 4′, 5′, and 6′.

Table 15 presents the results of minimizing *T* for Examples 5′ and 6′ by GSF. GSF has obtained much better solutions than single IGA (see Table 15), and the tardy orders is reduced by 2 and 4, respectively. Table 16 shows the tardy orders and the soft due dates. The soft due dates set with the smaller tardiness are much tighter, and thus further increase the hardness to obtain feasible solutions in minimizing *PT*. Nevertheless, GSF has made it!

With the soft due dates available, GSF is then applied to the three examples to minimize *PT*, where the objective function Eq. 6 is used. Table 17 shows the results. From this table, the following points can be observed:

(1) For Example 4′ (directly min. *PT* with the new due dates), a better solution ($PT = 2800.18$) is found.

(2) For Example 5′ (soft due dates are updated by T = 46.71), although the due dates are tighter, GSF has directly achieved the good feasible solution (PT = 3503.63)

(3) For Example 6′ (soft due dates are updated by T = 252.18), two runs of GSF have been carried out. In the first run (where $N_L = 30$), the final solution is infeasible, but only two orders (orders j5 and j14) have small negative start times, and the infeasible schedule with PT = 3971.46 is shown in Figure 7. At this time, we estimate that, if we conduct another GSF search (increase the local iterations of search, N_L, if necessary), then we may be sure to find a good feasible solution. In the second run (where $N_L = 50$), a good feasible solution has been indeed obtained, and the feasible schedule with PT = 3898.51 is shown in Figure 8.

(4) The results of Examples 4′, 5′, and 6′ solved by GSF verify the rigorous statement: under the soft due dates, there must be feasible solutions when minimizing the earliness related objectives (e.g., PT) with the backward assignment strategy. The search ability of GSF is indeed satisfactory.

To show the details of the convergence of GSF, we have added two tables and two plots (Figures 9a, b). Table 18 presents the results of in the Rough Search and Refining Search 1 of GSF with $N_L = 30$ for Example 6′; and Table 19 gives the comparison of the search processes of GSF with $N_L = 30$ and $N_L = 50$ for Example 6′. Table 19 and Figure 9 reveal several aspects of the search framework: (1) it is obvious that because the evolved solutions are added into restarted GA searches, E_{dev} decreases quickly in beginning global iterations, and then keep on a lower level with tiny waves; (2) the variation trend of the average CPU time (Av. CPU) over N_L IGA searches in a global iteration is similar to that of E_{dev}; (3) the average objective value (Av. PT) and the best one (Best PT) are persistently improving through the global iterations. In sum, the good solution added into a restarted GA speedup the evolution, and the global iterations keep improving the solution quality. Figure 9, drawn from Table 19, evidently indicates that, with the change of N_L from 30 to 50, GSF witnesses better average PT.

Conclusions and Future Work

This paper proposes an IGA. A comprehensive, but not very large set of position selection rules is first constructed according to the analysis of impact factors of scheduling objectives, and then suitable rules are selected for schedule synthesis procedure (decoding procedure). Compared with the OGA, the IGA enhances the performance by 10–20% for large-size instances in terms of solution quality, due to the appropriate rules selected for position selection in schedule synthesis.

In coping with infeasibility appearing during the search process, a penalty function is adopted. The penalty let infeasible solutions encountered in the search process constitute a bridge to find good feasible solutions.

To further enhance the solution quality, especially to solve the challenging scheduling problems with tight due dates, a novel GSF is then designed based on the concept of evolutionary gradient. This framework utilizes evolved solutions and evolutionary gradient to guide and control the global search. This makes the algorithm avoid repetitious computation that does not improve the solutions, and guarantees higher convergence speed and more robustness.

Case study shows that the improvement of the algorithm makes it more effective and efficient for large-size MMSP. In solving the challenging multi-stage scheduling problems with tight due dates, the GSF demonstrates good search ability.

In fact, the GSF can be easily applied to other combinatorial problems, e.g., the single-stage process scheduling problems.[30,31] Compared with the long-time TS which is frequently used to find the best solution for large-scale combinatorial problems, the GSF spends much shorter search time but finds much better solutions.

This framework can be easily transformed into a synchronous parallel version, because the synchronization and communication, which are the difficult issues in parallel computing, are already available. The further interesting work is to design an asynchronous parallel version of the framework.

The soft copy of the problem data and the package for Gantt charts of schedules, as well as other sources are available upon request from the authors.

Acknowledgments

The authors gratefully acknowledge the financial support from Hong Kong RGC grants 614307 and 614005. They also thank the three anonymous referees for their constructive comments, which are helpful for the improvement of this article.

Notation

Abbreviations

CP constraints	= changeover constraints and processing constraints
FSSP	= flow shop scheduling problem
GA	= genetic algorithm
GSF	= global search framework
HFS	= hybrid flow shop
IGA	= improved genetic algorithm
MILP	= mixed-integer linear programming
MINLP	= mixed-integer nonlinear programming
MMSP	= multi-stage multi-product scheduling problem
OGA	= original genetic algorithm
SMSP	= single-stage multi-product scheduling problem
TSP	= traveling salesman problem
UIS	= unlimited intermediate storage
SA	= simulated annealing
TS	= tabu search

Indices

g	= global iterations
i, j	= different customer orders, i, j 1, 2, ... , N
k	= different process stages, k 1, 2, ... , M
l	= local iterations, l 1, 2, ... , N_L
t	= iterations in GA
u	= different processing units, u 1, 2, ... , M_T

Sets

O	= a set of orders (with N orders)
U	= a set of units (with M_T units)

Parameters

c_{ij}	= changeover time when order i changeover to order j (not unit dependant)
CPU time	= computational time
C_r	= crossover rate in GA
dev. from best	= deviation from the best objective value
Diff1 (%)	= 100(objective value by MILP-objective value by OGA)/(objective value by OGA)
Diff2 (%)	= 100(objective value by OGA-objective value by IGA)/(objective value by IGA)
d_j	= due date, the completion date of order j promised to the customer

d_j' =	soft due date of order j
M =	the number of process stages
M_r =	mutation rate in GA
$msize$ =	the number of the chromosomes that are selected to mutate
M_T =	the total number of the process units
N =	the number of orders
N_L =	the number of IGA tests in a local search of GFS
N_{LL} =	the lower bound of N_L
N_{LU} =	the upper bound of N_L
or_j =	order release time, the earliest time at which order j can start its processing
p_{ju} =	the process time order j on unit u
$popsize$ =	the number of chromosomes in the initial generation of GA
ur_u =	unit release time, the time at which unit u can get ready
ut_u =	unit setup time of u
w =	penalty weight coefficient
$xsize$ =	the number of the chromosomes that are selected to crossover

Variables and functions

C_j =	completion time of order j
C_{max} =	makespan of a schedule
E =	total earliness
E_j =	earliness of order j
E_{dev} =	the mean deviation of N_L IGA tests
F =	total flow time
$f(E_{dev})$ =	the variable number of N_L depending on E_{dev}
$f(P)$ =	objective value of P
JP_i =	order time pointers
P =	a chromosome in GA for MMSP, $P = (P_1, P_2, \ldots, P_M)$, $P_j = (\pi_1, \pi_2, \pi_3, \ldots, \pi_N)$ is the segment of P at stage j
PR_j =	process time of order j through all stages
PT =	total generalized process time
π =	a permutation $(\pi_1, \pi_2, \ldots, \pi_N)$ in set O
RD_l =	the relative deviation of test l
S_j =	start time of order j
S_{min} =	the earliest start time of a schedule
T =	total tardiness
T_j =	tardiness of order j
UP_u =	unit time pointers

Literature Cited

1. Pinedo M. *Scheduling: Theory, Algorithms, And Systems*, 2nd ed. Upper Saddle, NJ: Prentice Hall, 2002.
2. Pinto JM, Grossmann IE. A continuous time mixed integer linear programming model for short term scheduling of multistage batch plants. *Ind Eng Chem Res*. 1995;34:3037–3051.
3. Pinto JM, Grossmann IE. A continuous time MILP model for short term scheduling of batch plants with pre-ordering constraints. *Comput Chem Eng*. 1996;20:1197–1202.
4. Hui CW, Gupta A, Meulen H. A novel MILP formulation for short term scheduling of multistage multi-products batch plants with sequence-dependent constraints. *Comput Chem Eng*. 2000;24:1611– 1617.
5. Mendez CA, Henning GP, Cerda J. An MILP continuous-time approach to short-term scheduling of resource-constrained multistage flowshop batch facilities. *Comput Chem Eng*. 2001;25:701–711.
6. Gupta S, Karimi IA. Scheduling a two-stage multiproduct process with limited product shelf life in intermediate storage. *Ind Eng Chem Res*. 2003;42:490–508.
7. Gupta S, Karimi IA. An improved MILP formulation for scheduling multiproduct, multistage batch plants. *Ind Eng Chem Res*. 2003; 42:2365–2380.
8. Castro PM, Grossmann IE. New continuous-time MILP model for the short-term scheduling of multistage batch plants. *Ind Eng Chem Res*. 2005;44:9175–9190.
9. Castro PM, Grossmann IE, Novais AQ. Two new continuous-time models for the scheduling of multistage batch plants with sequence dependent changeovers. *Ind Eng Chem Res*. 2006;45:6210–6226.
10. Liu Y, Karimi IA. Novel continuous-time formulations for scheduling multi-stage batch plants with identical parallel units. *Comput Chem Eng*. 2007;31:1671–1693.
11. Liu Y, Karimi IA. Scheduling multistage, multiproduct batch plants with nonidentical parallel units and unlimited intermediate storage. *Chem Eng Sci*. 2007;62:1549–1566.
12. Pinto JM, Turkay A, Bolio B, Grossmann IE. STBS: a continuous-time MILP optimization for short-term scheduling of batch plants. *Comput Chem Eng*. 1998;22:1297–1308.
13. Hui CW, Gupta A. A bi-index continuous time MILP model for short-term scheduling of single-stage multi-product batch plants with parallel line. *Ind Eng Chem Res*. 2001;40:5960–5967.
14. Harjunkoski I, Grossmann IE. Decomposition techniques for multi-stage scheduling problems using mixed-integer and constraint programming methods. *Comput Chem Eng*. 2002;26:1533–1552.
15. Maravelias CT. A decomposition framework for the scheduling of single- and multi-stage processes. *Comput Chem Eng*. 2006;30:407–420.
16. Wang K, Lohl T, Stobbe M, Engell S. A genetic algorithm for online-scheduling of a multiproduct polymer batch plant. *Comput Chem Eng*. 2000;24:393–400.
17. Ruiz R, Vazquez-Rodriguez JA. The hybrid flow shop scheduling problem. *Eur J Oper Res*. 2009, In press.
18. Chen C, Neppalli RV, Aljaber N. Genetic algorithms applied to the continuous flow shop problem. *Comput Ind Eng*. 1996;30:919–929.
19. Ruiz R, Maroto C. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *Eur J Oper Res*. 2006;169:781–800.
20. Tandon M, Cummings PT, Levan MD. Scheduling of multiple product on parallel units with tardiness penalties using simulated annealing. *Comput Chem Eng*. 1995;19:1069–1076.
21. Low CY. Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines. *Comput Oper Res*. 2005; 32:2013–2025.
22. Allahverdi A, Al-Anzi FS. Scheduling multi-stage parallel-processor services to minimize average response time. *J Oper Res Soc*. 2006; 57:101–110.
23. Logendran R, deSzoeke P, Barnard F. Sequence-dependent group scheduling problems in flexible flow shops. *Int J Prod Econ*. 2006; 102:66–86.
24. Wang X, Tang L. A tabu search heuristic for the hybrid flowshop scheduling with finite intermediate buffers. *Comput Oper Res*. 2009;36:907–918.
25. He Y, Hui CW. Genetic algorithm for large-size multi-stage batch plant scheduling. *Chem Eng Sci*. 2007;62:1504–1527.
26. He Y, Hui CW. Automatic rule combination approach for single-stage process scheduling problems. *AIChE J*. 2007;53:2026–2047.
27. Linn R, Zhang W. Hybrid flow shop scheduling: a survey. *Comput Ind Eng*. 1999;37:57–61.
28. Brooke A, Kendrick D, Meeraus A. *GAMS—A User's Guide (Release 2.25)*. San Francisco, CA: The Scientific Press, 1992.
29. Rajendran C, Ziegler H. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *Eur J Oper Res*. 2004;155:426–438.
30. He Y. Research on Meta-Heuristic Methods for Large-Scale Complex Scheduling in Process Industry. Ph.D. Dissertation, Hong Kong University of Science and Technology, Hong Kong, 2007.
31. He Y, Hui CW. *Meta-Heuristics for Large-Scale Process Scheduling*. Saarbrücken, Germany: VDM Verlag, 2009.

Manuscript received July 27, 2009, and revision received Nov. 4, 2009.