Computers and Chemical Engineering 32 (2008) 3067–3083

Contents lists available at ScienceDirect

Computers and Chemical Engineering

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

A rule-based genetic algorithm for the scheduling of single-stage multi-product batch plants with parallel units

Yaohua He, Chi-Wai Hui*

Chemical Engineering Department, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong, PR China

ARTICLE INFO

Article history:
Received 3 November 2005
Received in revised form 4 May 2007
Accepted 28 April 2008
Available online 4 May 2008

Keywords:
Parallel unit scheduling
Mixed-integer linear programming
Random search
Genetic algorithm
Heuristic rule

ABSTRACT

This paper presents a heuristic rule-based genetic algorithm (GA) for large-size single-stage multi-product scheduling problems (SMSP) in batch plants with parallel units. SMSP have been widely studied by the researchers. Most of them used mixed-integer linear programming (MILP) formulation to solve the problems. With the problem size increasing, the computational effort of MILP increases greatly. Therefore, it is very difficult for MILP to obtain acceptable solutions to large-size problems within reasonable time. To solve large-size problems, the preferred method in industry is the use of scheduling rules. However, due to the constraints in SMSP, the simple rule-based method may not guarantee the feasibility and quality of the solution. In this study, a random search based on heuristic rules was proposed first. Through exploring a set of random solutions, better feasible solutions can be achieved. To improve the quality of the random solutions, a genetic algorithm-based on heuristic rules has been proposed. The heuristic rules play a very important role in cutting down the solution space and reducing the search time. Through comparative study, the proposed method demonstrates promising performance in solving large-size SMSP.

1. Introduction

Scheduling plays important roles in increasing the efficiency of any production systems. In contrast to the long term concerns with scheduling in discrete parts manufacturing industry, the attention devoted to computer-aided scheduling methodology in the process industry is much more recent, beginning in the middle 1970's (Ku, Rajagopalan, & Karimi, 1987; Reklaitis, 1982). Process scheduling shows much more complexity than discrete machine scheduling. Research on batch and continuous process scheduling has received great attention from academia and industry in the past two decades.

General mathematical formulations for process scheduling are either based on the state task network (STN) (Kondili, Pantelides, & Sargent, 1993) or resource task network (RTN) (Pantelides, 1993) process representations. STN or RTN-based formulations can be applied to any network-represented processes from batch to continuous, consisting of fixed or variable duration tasks. The major difference between STN and RTN is that the STN treats equipment resources implicitly, while the RTN treats them explicitly. Earlier formulations use a discrete-time representation and may involve several thousands of binary variables in order to handle a sufficiently fine discretization that closely matches the exact problem data. As a consequence, when solving large-size problems, problem data is usually rounded to maintain problem tractability. Overall, discrete-time mixed-integer linear programming (MILP) formulations are usually very tight and perform well for a variety of objective functions, even makespan minimization (Maravelias & Grossmann, 2003b). Continuous-time formulations began to appear in the last decade, and have received much attention recently. In continuous-time formulations one needs to specify the number of time points that compose the time grids, depending on whether the formulation uses a single time grid (Castro, Barbosa-Povoa, Matos, & Novais, 2004, Castro, Barbosa-Povoa, & Novais, 2005; Maravelias & Grossmann, 2003a), or one for each equipment resource (Giannelos & Georgiadi, 2002).

Single-stage multi-product scheduling problems (SMSP) in batch plants with parallel units are a typical kind of scheduling problems in process industry. An extension of SMSP is the multi-stage multi-product scheduling problems (MMSP) in batch plants

Abbreviations: STN, state task network; RTN, resource task network; SMSP, single-stage multi-product scheduling problem; MMSP, multi-stage multi-product scheduling problem; MP, mathematical programming; MILP, mixed-integer linear programming; CP, constrain-programming; SA, simulated annealing; TS, tabu search; GA, genetic algorithm; CP constraints, changeover constraints and processing constraints; EDD, earliest due date; FAU, first available unit; SCT, shortest changeover time; SPT, shortest process time; EST, earliest start time; SPsPT, shortest possible start time + process time; SCPT, shortest changeover + process time; PMX, partial-mapped crossover; RS_Rn, RS combined with *Rule n*; GA_Rn, GA combined with *Rule n*.

* Corresponding author. Tel.: +852 23587137; fax: +852 23580054.
E-mail address: kehui@ust.hk (C.-W. Hui).

0098-1354/$ – see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compchemeng.2008.04.008

Nomenclature

Indices

i, j	different customer orders
t	iterations of the algorithm
u	processing units

Sets

O	a set of orders (with N orders)
U	a set of units (with M units)

Parameters

c_{ij}	changeover time when order i changeover to order j (not unit dependant)
CPU, CPU time	computational time
C_r	crossover rate
d_j	due date: the committed shipping or completion date of order j (the date the order is promised to the customer)
N	the number of orders
msize	the number of the chromosomes that are selected to mutate
M	the number of units
M_r	mutation rate
or_j	order release time: the earliest time at which order j can start its processing
p_{ju}	order process time
ur_u	unit release time: the earliest time at which unit u can get ready
popsize	the number of chromosomes in the initial generation
xsize	the number of the chromosomes that are selected to crossover
α, β	weight coefficient in the compound scheduling objective

Variables and functions

C_j	completion time of order j
$C_{\max}$	makespan
E	total earliness
E_j	earliness of order j
$f(\pi)$	objective value of π
T	total tardiness
TC	compound objective of makespan and total tardiness
T_j	tardiness of order j
π	a permutation $(\pi_1, \pi_2, \ldots, \pi_N)$ in set O, $\pi_i \in \{1, 2, 3, \ldots, N\}$, $i = 1, 2, 3, \ldots, N$.

with parallel units. There is a large body of literature on these two kinds of problems.

Pinto and Grossmann (1995) presented a continuous-time MILP model for MMSP. They used the concept of parallel time coordinates for units and tasks. Furthermore, Pinto and Grossmann (1996) proposed an alternative model in which the pre-ordering of orders was imposed explicitly, by applying an alternative representation of the time slots for the units. This resulted in a significant reduction in computational time. Cerda, Henning, and Grossmann (1997) proposed a continuous-time MILP model for SMSP. They used tri-index decision variables as well as the concept of predecessor and successor to describe the order assignment to various production units while taking into account sequence-dependent changeover constraints. To deal with large-size problems, they proposed heuristics, such as the order pre-ordering, to reduce the number of feasible predecessors for each order. On the basis of the notation of time slot, Karimi and McDonald (1997) proposed two models for parallel semicontinuous processes considering the sequence-dependent setup times, orders, and their corresponding due dates in order to minimize the inventory. The major advantage of the formulation is that it can incorporate fixed time events such as due dates while using continuous representation of time.

Hui and Gupta (2001) presented a general formulation for SMSP. The proposed formulation applies three sets of bi-index variables to handle order sequence-dependent constraints either with or without imposing pre-ordering heuristics. The main advantage of this formulation compared to the other previously proposed formulations is the significant reduction in the number of binary variables and consequently shortening the solution time. The authors claimed their method suitable for handling large-size industrial problems.

Chen, Liu, Feng, and Shao (2002) developed a short-term scheduling model for SMSP using the continuous-time representation and the notation of time slot. When the model was developed, the allocation of orders and units to time slots was represented by two sets of binary variables. The MILP model not only involved fewer binary variables than any other models based on the notation of time slot (Pinto & Grossmann, 1998) but also could be used to optimize several types of objective functions.

An important recent advance was the introduction by Sundaramoorthy and Karimi (2005) of a formulation without big M constraints that proved more efficient than other competing methods for both maximizing profit and minimizing makespan.

Castro and Grossmann (2006) presented a new RTN-based continuous-time MILP model for SMSP. It was based on the general formulation of Castro et al. (2004), but had one important difference: a different time grid was used for each unit instead of a single time grid for all events taking place. New constraints were presented that allowed the consideration of both release and due dates in a general way. The new formulation was shown to perform much better than other continuous-time MILP formulations and standalone constrain-programming models (Jain & Grossmann, 2001), and was comparable to the hybrid MILP/CP algorithm of Maravelias and Grossmann (2004) on a set of example problems where the objective was the minimization of total cost, or the minimization of total earliness. However, this model did not consider changeovers between products. Hence, Castro, Grossmann, and Novais (2006) later presented two new multiple-time-grid continuous-time formulations for SMSP and MMSP, where equipment units are subject to sequence-dependent changeovers, and product orders are 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 and Grossmann (2006) is not the best efficient mathematical programming method. The CP method and the discrete-time formulation were found to be the best mathematical programming methods (Castro & Grossmann, 2005). However, these two methods have the disadvantage of considering integer data and show a rapid decrease in performance with an increase in problem complexity.

Heuristic rules are a widely used to cut down the size of MILP models. Reklaitis and Mokus (1995) proposed a non-uniform time discretization model, in which binary variables were applied to represent the start and stop events for the various recipe tasks. They used the randomized heuristic approach (Mockus & Reklaitis, 1996) to reduce the number of binary variables. Pinto and Grossmann (1996) greatly cut down the size of the model (Pinto & Grossmann, 1995) for MMSP through imposing the pre-ordering constraints (heuristic rules) into it. The formulation of the simplified model involved only decisions of assignment of orders to units and timing of each processing task. However, the pre-ordering constraints affected the optimality of the scheduling model. Cerda et al. (1997) reduced the size of the scheduling model for SMSP by introducing some heuristic rules, so that the feasible predecessors of an order were pruned. These heuristic rules were also used by Mendez and Cerda (2000), Mendez, Henning, & Cerda (2000) and Hui, Gupta, and Meulen (2000). However, when these heuristic rules were used, the optimal solution could not be obtained, especially when makespan was minimized. The MILP model proposed by Chen et al. (2002) introduced some heuristic rules that could cut down the size of the model and had no effect on the optimality of the scheduling problem.

Further details on the above mentioned and other available approaches for short-term scheduling can be found in the recent review papers of Floudas and Lin (2004) and Mendez, Cerda, Grossmann, Harjunkoski & Fahl (2006). All of the above MILP methods focused on small-size problems. When the problem size increases linearly, the computational effort of MILP will increase greatly. It is very difficulty for MILP models to solve large-size problems. Although some researchers (Hui & Gupta, 2001; Pinto, Turkay, Bolio, & Grossmann, 1998) claimed their MILP models suitable for large-size problems, but the solutions to large-size problems were far from the optimality, even if the algorithm ran for very long time, e.g. several hours or over 10 h.

Meta-heuristic methods, such as genetic algorithm (GA), simulated annealing (SA) and tabu search (TS), although they cannot prove optimality, can provide optimal or near-optimal solutions within moderate or acceptable computational time. Therefore, meta-heuristic methods are more suitable for large-size problems. There has been a large body of literature on meta-heuristic methods for scheduling problems in discrete manufacturing systems; but only several papers on GA (Chen, Neppalli, & Aljaber, 1996; Jou, 2005; Kim, Jang, & Lee, 1996), SA (Das, Cummings, & Le Van, 1991; Ku & Karimi, 1991), list-based threshold-accepting algorithm (LBTA) (Lee, Vassiliadis, & Park, 2002) for process scheduling problems. Some authors believe that, due to the high complexity and constraints of process scheduling, meta-heuristic methods are not suitable for process scheduling problems. Their reason is that the original feasible solutions for evolution are not easy to be achieved.

In this paper, we present a genetic algorithm-based on heuristic rules for large-size SMSP. In our work, the size of the problems was enlarged, and the problems are first solved by MILP methods. And then a random search (RS) based on heuristic rules has been proposed. Subsequently, GA based on heuristic rules is proposed to evolve a set or random solutions. The proposed GA is able to handle SMSP subject to order release times and due dates, unit release times, both sequence- and unit-dependant changeovers and the cases with forbidden changeovers and processes. Furthermore, our proposed method does not require rounded problem data. The most advantage of this method is that it can solve the large-size problems to acceptable solutions within very shorter time in contrast to MILP models. The algorithms in this paper were implemented in C language and the computation tests were run on a computer with an Intel Pentium M 1500 MHz CPU and 768 MB of memory.

2. Problem definition

In SMSP, a fixed number of production units (forming a set of units: U) are available to process all customer orders (forming a set of orders: O). Assume the number of production units is M, and the number of customer orders is N. Each order involves a single product, requiring a single processing stage, has a predetermined due date, and can only be processed in a subset of the units available. The production units have different processing capacities. Hence, the process time of the same order is fixed and production unit dependent. A production unit processes only one order at a time. When one order changes over to another order, time is required for the preparation of the unit for the changeover. The changeover times are sequence-dependent or both sequence- and unit-dependant. Forbidden changeovers and processes, called CP constraints, may exist in the problem.

Time-based scheduling objectives are commonly considered in the literature because cost-based objectives usually can be surrogated by time-based objectives (Pinedo, 1995). Common time-based objectives are described by Pinedo (1995). Makespan, total tardiness and total earliness are three typical scheduling objectives to be minimized.

(1) *Makespan* ($C_{\max}$): The makespan, defined as:

$$C_{\max} = \max\{C_1, C_2, \ldots, C_N\} \tag{1}$$

is the completion time of the last order to leave the system, where C_j is the completion time of order j. A minimum makespan usually implies a high utilization of the units.

(2) *Total tardiness* (T): The total tardiness is the sum of the tardiness of all orders, defined as:

$$T = \sum_{j=1}^{\mathrm{N}} T_j \tag{2}$$

where T_j is the tardiness of order j: $T_{\mathrm{j}} = \max\{C_j - d_j, 0\}$, d_j is the due date of order j.

(3) *Total earliness* (E): The total earliness is the sum of the earliness of all orders, defined as:

$$E = \sum_{j=1}^{\mathrm{N}} E_j \tag{3}$$

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

The scheduling objective in this work is to minimize the makespan $C_{\max}$ or total tardiness T, depending on the due dates of the orders. If it is obvious that all the orders can be completed before their due dates, it makes no sense to minimize the total tardiness.

Just as done in the literature (Cerda et al., 1997; Hui & Gupta, 2001), when minimizing the makespan of Examples 1–3, the due dates of the orders are not considered as constrains. If due dates are considered as constraints when minimizing makespan, the complexity of MILP model increases. However, in our work, it is just an option, not an actual limitation, that the due dates are not considered as constraints when minimizing makespan. Actually, adding a penalty term in the objective function for the orders violating due dates enable the algorithm to find a good schedule without tardy orders, assuming that there are no tardy orders at optimum. In case of tardy orders at optimum, soft due dates (He & Hui, 2007a) should be used, which is determined by minimizing the tardiness, hence increasing computational effort. Therefore, we do not consider due dates as constrains when minimizing makespan in Examples 1–3.

(4) The minimization of the makespan and the minimization of the total tardiness can be conflicting objectives. If the due dates of the orders are loose, it is easy to find a schedule with zero total tardiness; but the makespan of the schedule may be still very long. To consider both makespan and total tardiness at the same time, we propose a new *compound objective* TC:

Table 1
Problem data of the 10-order/4-unit instance of Example 1

	Changeover times (c_{ij})										d_i	or_i	Process times (p_{iu})			
	j1	j2	j3	j4	j5	j6	j7	j8	j9	j10			u1	u2	u3	u4
i1	–	1.00	0.15	1.10	2.00	0.65	0.30	1.20	0.85	0.40	10	0	10.20	3.60	4.20	10.80
i2	1.80	–	1.10	1.30	1.40	0.90	0.20	1.20	0.40	0.30	22	0	7.20	10.50	4.50	4.50
i3	1.00	0.15	–	1.20	1.50	2.10	0.30	1.80	1.60	0.20	25	0	6.25	5.20	5.50	5.00
i4	1.20	0.02	0.10	–	0.05	1.60	1.20	2.00	1.20	0.50	20	0	11.20	13.60	15.40	12.00
i5	0.10	0.20	0.30	0.30	–	0.70	0.90	0.60	1.00	0.90	28	0	9.00	8.40	4.50	3.20
i6	1.40	0.80	0.30	0.70	2.00	–	0.90	1.20	1.20	1.60	30	0	9.60	4.00	10.80	5.50
i7	1.20	1.80	1.30	0.90	0.85	0.80	–	0.45	1.20	1.30	17	0	7.60	6.00	3.00	6.40
i8	1.30	1.40	1.50	1.40	1.20	1.30	1.65	–	1.30	0.80	23	0	14.00	14.70	16.80	16.80
i9	2.10	2.00	1.25	1.35	1.45	0.80	1.60	0.80	–	0.65	30	0	4.80	3.00	6.30	3.60
i10	1.50	1.20	0.60	0.75	0.50	0.40	0.90	0.60	0.70	–	30	0	7.80	5.70	4.80	7.20
d_i: Due date; or_i: order release time; ur_u: unit release time												ur_u	0	0	0	0

$$\mathrm{TC} = \alpha T + \beta C_{\max} \tag{4}$$

where T is the total tardiness, and $C_{\max}$ is the makespan, and α and β are weight coefficients. We may let $\alpha = \beta = 1$. In Examples 4 and 5, this compound objective is used in order to satisfy the order due dates.

A small-size instance of Example 1, the problem data of which is presented in Table 1, will be used to illustrate our approach. Example 1 originated from (Cerda et al., 1997) and was later studied by Hui and Gupta (2001) and Chen et al. (2002).

3. MILP model and its difficulties

For SMSP, Cerda et al. (1997) did the original work with MILP model, later Karimi and McDonald (1997), Hui and Gupta (2001) and Chen et al. (2002) presented their modified MILP models. All these works focused small-size problems, and the later improvements were marginal, not substantial, especially for large-size problems. So the MILP model by Hui and Gupta (2001), a bi-index MILP model based on continuous-time formulation, is representative, maybe not very efficient. In the literature (Cerda et al., 1997; Hui & Gupta, 2001), a pre-ordering heuristic is adopted to reduce search space—orders assigned to a unit are processed in a sequence of increasing due dates. By this way, search time is reduced, but the optimum of the original problem cannot be guaranteed. That is to say, even if an optimal solution to the revised model is achieved, it is not the optimum of the original problem.

In Table 2, the basic model is the bi-index model by Hui and Gupta (2001), and the revised model is based on the basic one, but including the pre-ordering heuristic. These two models is formulated in GAMS and solved by the solver OSL on a PC with an Intel Pentium M 1500 MHz CPU and 768 MB of memory. Fig. 1 shows the Gantt chart of a schedule of the 10-order/4-unit small instance by the revised MILP.

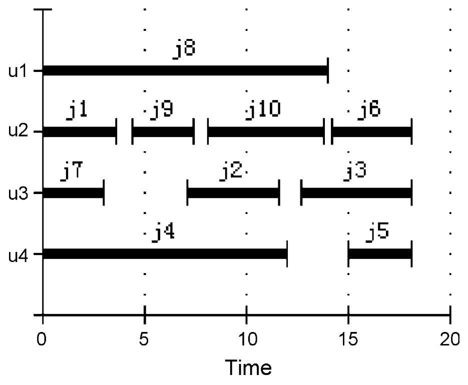

Fig. 1. A schedule of the 10-order/4-unit small instance by MILP ($C_{\max}$ = 18.20).

As for the MILP models for solving the small sample instance and large instances later, our computational experiments show the following phenomena or difficulties:

(1) The MILP model has to run for a certain number of iterations before it finds a feasible solution. For instance, in solving the small sample instance, at iteration 537 by the basic model, and at iteration 411 by the revised model, the algorithm showed "No integer solution yet". In solving more large-size instances by MILP, it takes much more iterations (or CPU time) to get a feasible solution.
(2) At the early stage of the computation, solution quality increases quickly. However, after running for some time, it often takes relatively long time to get another better solution. In solving the small-size instance, even at iteration 2,000,000 which takes about 10 min, the best solution with a makespan 17.35 (achieved by the subsequent random search and genetic algorithm) is not achieved by the basic model. With the problem size

Table 2
Results of the 10-order instance of Example 1by MILP (min $C_{\max}$)

Basic model				Revised model (with pre-ordering heuristic)			
Iterations	Nodes	CPU time (s)	$C_{\max}$	Iterations	Nodes	CPU time (s)	$C_{\max}$
537	No integer solution yet			411	No integer solution yet		
1000	49	0.23	30.05	1000	66	0.24	26.00
2000	109	0.43	21.70	2000	129	0.44	20.10
10000	624	1.90	21.40	10000	694	1.92	21.80
20000	1231	4.43	21.40	20000	1391	4.36	18.20
100000	6133	19.28	20.25	100000	7318	19.09	18.20
200000	12467	45.92	20.25	200000	14823	39.05	18.20
1000000	62314	275.08	19.50	853528	71358	190.48	18.20
2000000	121124	646.24	18.45				

Table 3
Scheduling by EDD

Scheduling rule	Earliest due date (EDD)									
Order *i*	i1	i7	i4	i2	i8	i3	i5	i6	i9	i10
d_i	10	17	20	22	23	25	28	30	30	30
Order sequence	1,	7,	4,	2,	8,	3,	5,	6,	9,	10
Unit selection rules	(a) *First available unit* (FAU); (b) *shortest processing time (SPT)*									

increasing, the search time to find a "good enough" solution is getting unbearable.

(3) Theoretically, MILP can find the optimal solution to a problem if given enough time. However, even if the best solution is found, a lot effort or time may be required to prove the optimality. For some small-size problems, the optimum may be found within a small number of iterations. In solving the small sample instance by the revised model, the makespan 18.20 had been obtained at iteration 20,000, but the optimum had not been affirmed until iteration 853,528. As the problem size increases, the search time to affirm the optimum is getting more unbearable.
(4) After the basic model has been established, adding some constraints (e.g. the pre-ordering heuristic) or changing the parameters in the existing constraints (e.g. the Big *M*) makes it possible to get much better solutions within shorter time. To do this, a "trial and error" method has to be used. A lot of time is needed to find the suitable parameters in the constraints. Hence, simulation experiments are required to tune the MILP models.

4. The use of scheduling rules

Due to difficulties of MILP for large-size scheduling problems, the preferred approach in industry is the use of scheduling rules, such as the shortest processing time first (SPT) rule and the earliest due date first (EDD) rule. According to a scheduling rule, the customer orders are sequenced in decreasing priority order, and then one by one assigned to the units according to a unit selection rule, such as the first available unit (FAU). FAU is usually used by practitioners. Table 3 shows that the 10 orders are sequenced according to increasing due dates, and Fig. 2 shows the schedules obtained, the makespans (25.90 and 27.20) of which is longer than 18.20 in Fig. 1.

If CP constraints do not exist in the problem, by using scheduling rules, one feasible solution can be acquired. Unfortunately, if CP constraints exist in the problem, only by using scheduling rules, feasible solution may not be found. Moreover, the effective rules change with the scheduling objective and the problem conditions.

5. Random search based on heuristic rules

In order to obtain much better solutions, a random search (RS) could be adopted: the customer orders are randomly sequenced, and then one by one assigned to the production units according to a unit selection rule. Through RS, a number of feasible solutions can be obtained, among which much better solutions may be found.

5.1. Representation of a random solution

Natural numbers 1, 2, 3, . . ., *N* are used to denote *N* orders. A random order sequence $\pi = (\pi_1, \pi_2, \pi_3, \ldots, \pi_N)$ is produced, $\pi_i \in \{1, 2, 3, \ldots, N\}$, $i = 1, 2, 3, \ldots, N$. And then, from π_1 to π_N, one by one, each order will be assigned over the *M* units according to a certain heuristic rule (unit selection rule). As a result, considering the constraints given, if all of the orders are able to be assigned to the units, then π stands for a feasible solution, and the objective value is $f(\pi)$; otherwise, π is infeasible. In fact, an order sequence is called a chromosome in later genetic algorithm. As the scheduling objective is to minimize makespan or total tardiness, so for makespan:

$$f(\pi) = C_{\max} = \max\{C_1, C_2, \ldots, C_N\} \tag{5}$$

where C_j is the completion time of order *j*; for total tardiness:

$$f(\pi) = T = \sum_{j=1}^{N} T_j \tag{6}$$

where T_j is the tardiness of order *j*: $T_j = \max\{C_j - d_j, 0\}$.

5.2. Heuristic rules

In the other work (He & Hui, 2007b), we have summarized seven unit selection rules for minimization of the makespan related objectives in SMSP, through analyzing impact factors on the objective. Table 4 presents the rules. These behave differently in scheduling the orders.

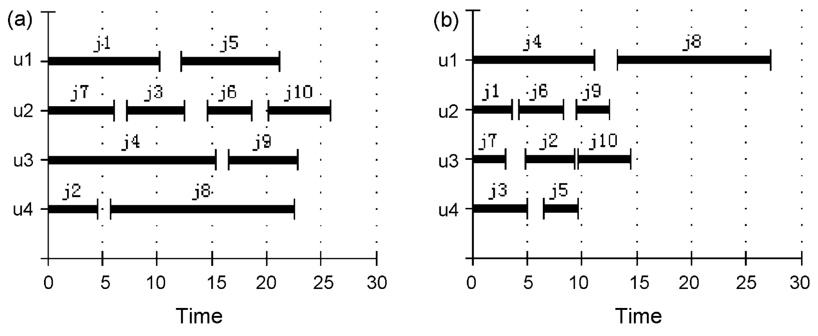

Fig. 2. Two schedules from an order sequence using EDD. (a) Select units by FAU ($C_{\max} = 25.90$); (b) select units by SPT ($C_{\max} = 27.20$).

Table 4
Seven rules for minimization of the makespan related objectives in SMSP

Rule	Detailed content	Shortened form	Factors involved	Simple or compound
Rule 1	Assign the order on the unit that makes the order's possible start time be as early as possible, that is, assign the order on the first available unit (FAU)	Earliest possible start time, FAU	Possible start time (PsT)	Simple
Rule 2	Assign the order on the unit that makes the order's changeover time on the unit be the shortest	Shortest changeover time, SCT	Changeover time (CT)	Simple
Rule 3	Assign the order on the unit that makes the order's process time on the unit be the shortest	Shortest process time, SPT	Process time (PT)	Simple
Rule 4	Assign the order on the unit that makes the order's start time be as early as possible	earliest start time, EST	Possible start time and changeover time	Compound
Rule 5	Assign the order on the unit that makes the sum of the order's possible start time and process time on the unit be the shortest	Shortest possible start time + process time, SPsPT	Possible start time and process time	Compound
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	Shortest changeover time + process time, SCPT	Changeover time and process time	Compound
Rule 7	Assign the order on the unit that makes the order's completion time be as early as possible	Earliest completion time, ECT	Possible start time, changeover time and process time	Compound

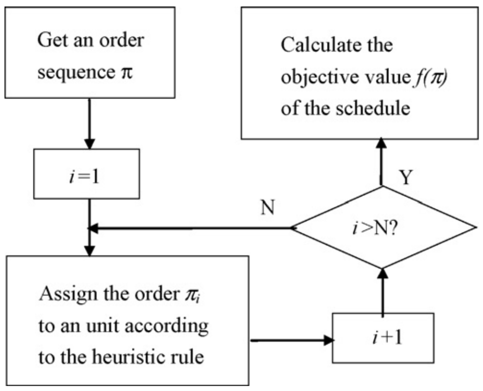
Get an order sequence π
i=1
Assign the order π_i to an unit according to the heuristic rule
i+1
i>N?
N
Y
Calculate the objective value $f(\pi)$ of the schedule

Fig. 3. Schedule synthesis by a heuristic rule.

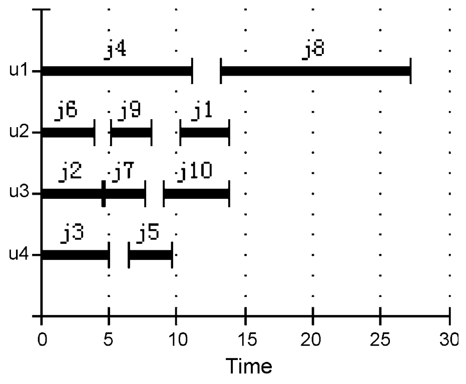

Fig. 4. A schedule from a random order sequence (synthesized with SPT).

5.3. Schedule synthesis by heuristic rules

The procedure of schedule synthesis by a heuristic rule could be illustrated as Fig. 3. The selected heuristic rule (e.g. SPT) is the criterion to assign every order in the order sequence to a unit. Assume that a random order sequence $\pi = (3, 2, 7, 6, 4, 5, 9, 10, 1, 8)$ is to be synthesized into a schedule according to SPT. In Table 5, the shortest process times are shown in italic. Fig. 4 shows the schedule synthesized. According to Table 5, it is easy to explain why the orders are assigned as shown in Fig. 4. For example, order 3 is assigned to unit 4, for its process time on unit 4 is the shortest.

Table 6 presents the performance difference between the rules. The random order sequence is scheduled by all the seven rules, resulting in different solutions, and Fig. 5 shows the different schedules. The situation for the order sequence by EDD is the same. Rule 5 and Rule 7 demonstrate good performance in minimizing makepsan.

Table 5
Process times of the orders

Order	Process times (p_{iu})			
	u1	u2	u3	u4
i1	10.20	***3.60***	4.20	10.80
i2	7.20	10.50	***4.50***	4.50
i3	6.25	5.20	5.50	***5.00***
i4	***11.20***	13.60	15.40	12.00
i5	9.00	8.40	4.50	*3.20*
i6	9.60	***4.00***	10.80	5.50
i7	7.60	6.00	***3.00***	6.40
i8	***14.00***	14.70	16.80	16.80
i9	4.80	***3.00***	6.30	3.60
i10	7.80	5.70	***4.80***	7.20
π	3, 2, 7, 6, 4, 5, 9, 10, 1, 8			

Table 6
Two order sequences scheduled by different rules

Random π	3, 2, 7, 6, 4, 5, 9, 10, 1, 8						
Rule used	*Rule 1*	*Rule 2*	*Rule 3*	*Rule 4*	*Rule 5*	*Rule 6*	*Rule 7*
Makespan $C_{\max}$	27.45	32.35	27.20	27.45	**24.80**	29.80	**24.80**
Total tardiness T	1.60	12.35	0.00	1.60	0.00	0.00	0.00
π by EDD	1, 7, 4, 2, 8, 3, 5, 6, 9, 10						
Rule used	*Rule 1*	*Rule 2*	*Rule 3*	*Rule 4*	*Rule 5*	*Rule 6*	*Rule 7*
Makespan $C_{\max}$	25.90	30.75	27.20	25.90	**19.50**	29.80	**19.50**
Total tardiness T	0.00	0.00	0.00	0.00	0.00	0.00	0.00

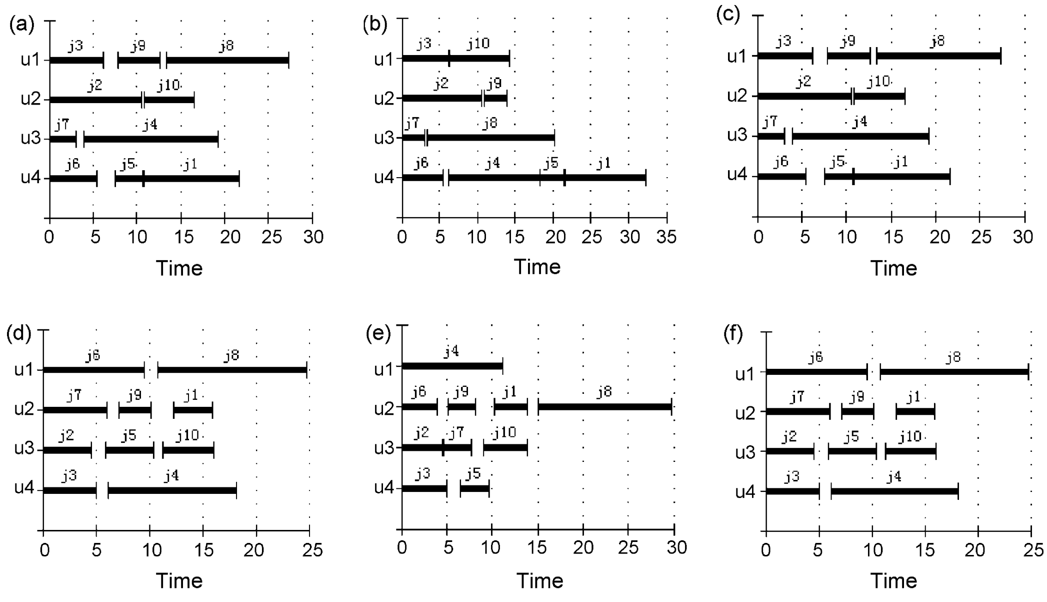

Fig. 5. Schedules from a random order sequence (synthesized with different rules). (a) With Rule 1 (FAU); (b) With Rule 2 (SCT); (c) With Rule 4 (EST); (d) With Rule 5 (SPsPT); (e) With Rule 6 (SCPT); (f) With Rule 7 (ECT).

5.4. *Procedure of the random search*

The procedure of the random search as shown in Fig. 6, can be easily understood. The number of random feasible solutions searched is *popsize*, which is determined according to the problem size.

5.5. *Computational results by the random search with different rules*

Table 7 presents the computational results of the small sample instance by the random search combined with different heuristic rules. For each method, 10 tests of computation were performed. The makespan of the current best schedule is 17.35. From Table 7, it can be noticed that: (1) *Rule 5*, *Rule 7* and *Rule 4* are good quality rules that enable RS to obtain short makespan. (2) *Rule 1*, *Rule 2*, *Rule 6 and Rule 3* are poor quality rules that make the RS obtain longer makespan. *Rule 3* is the worst rule.

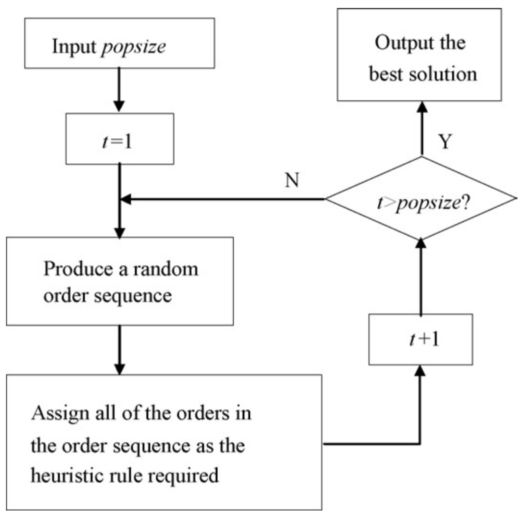
Input *popsize*
t=1
Produce a random order sequence
Assign all of the orders in the order sequence as the heuristic rule required
t+1
t>*popsize*?
N
Y
Output the best solution

Fig. 6. Flow chart of random search.

6. Genetic algorithm-based on heuristic rules

To evolve a set of random solutions, a heuristic rule-based GA is proposed in this paper. The proposed method combines GA with a unit selection rule for specific scheduling objective. At the beginning of GA, a number of random solutions are generated. Through evolutionary mechanism of GA, optimal or near-optimal solutions can be achieved.

6.1. *Components of genetic algorithm*

6.1.1. *Representation of solutions*

In solving scheduling problems by GA, first task is to represent a solution of the problem as a chromosome. Permutation-based representation is adopted in this paper. As stated in Section 5.1, natural numbers 1, 2, 3, . . ., *N* are used to denote *N* orders. A order sequence $\pi = (\pi_1, \pi_2, \pi_3, \ldots, \pi_N)$, $\pi_i \in \{1, 2, 3, \ldots, N\}$, is called a chromosome in GA. A sample chromosome of 10-order/4-unit instance is shown in Fig. 7.

6.1.2. *Generation*

At the beginning of GA, an initial generation of chromosomes are randomly generated. Assume the number of chromosomes in the initial generation is *popsize*. At each iteration of GA, a

Table 7
Results of 10-order/4-unit instance in Example 1 by RS (min $C_{\max}$)

Method	Best	Mean $C_{\max}$	Mean CPU time (s)	Rule quality
RS_R1 (FAU)[a]	18.15	19.58	<0.055	Poor
RS_R2 (SCP)	18.30	20.58	<0.055	Poor
RS_R3 (SPT)	26.60	26.60	<0.055	Poor
RS_R4 (EST)	17.75	18.91	<0.055	Good
RS_R5 (SPsPT)	**17.35**	17.89	<0.055	Good
RS_R6 (SCPT)	18.45	19.34	<0.055	Poor
RS_R7(ECT)	**17.35**	17.79	<0.055	Good

[a] RS_Rn stands for RS combined with Rule *n*.

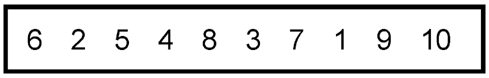

Fig. 7. A sample permutation-based chromosome.

new generation will be produced by crossover, mutation and selection.

6.1.3. *Selection*

Selection is the process to select most of the better chromosomes in each generation to crossover, and the rest or some of the rest to mutate. Although the roulette-wheel selection (Goldberg, 1989) is one commonly used technique, but *tournament* method (Goldberg & Deb, 1991) is adopted in our GA, because in the *tournament* method objective values can be used as selection criteria, not always the fitness as in roulette-wheel selection. Assume the number of the chromosomes that are selected to crossover is *xsize*; the number of the chromosomes that are selected to mutate is *msize*. We may let $xsize + msize = popsize$.

The ratio $C_r = xsize/popsize$ is called crossover rate, often $C_r \in [0.5, 0.9]$; and the ratio $M_r = xsize/popsize$ called mutation rate, often $M_r \in [0.1, 0.3]$. So $C_r + M_r = 1$. C_r and M_r are two important parameters that influence the convergent performance of GA. In general, if M_r increases, GA will be convergent to the final solution slowly, so that GA has more chances to find out better solutions. But if M_r is too large, GA tends to be like random search (Goldberg, 1989). We let $C_r = 0.8$ and $M_r = 0.2$ in this work.

6.1.4. *Crossover*

There are several methods to crossover (Poon & Carter, 1995). Partial-mapped crossover (PMX) is adopted in the proposed GA, as shown in Fig. 8.

6.1.5. *Mutation*

Mutation is the operation to make some genes of a chromosome changed. There are also a number of methods to mutate, such as insertion, swap and reversion, see Fig. 9. Reversion is employed in this paper

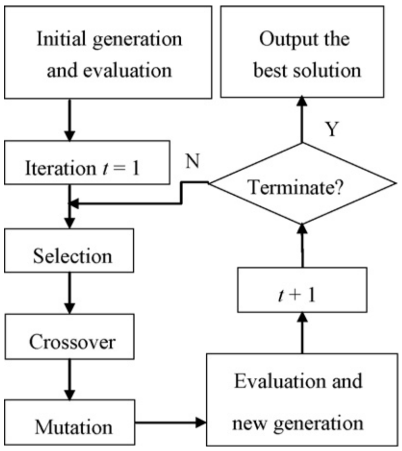

Fig. 10. Flow chart of GA.

6.1.6. *Termination condition*

The algorithm terminates when no further improvement on the solutions exists (named *until-no-improvement*). With the algorithm running on, the difference among the chromosomes will tend to be null. That means no further improvement on the solutions exists, the algorithm should terminate. The termination condition for GA in this work 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 0.001.

6.2. *Procedure of genetic algorithm*

Fig. 10 is the illustration of the whole GA procedure that consists of the following steps:

- *Step 1*. Initialization and initial evaluation: produce an initial generation in which there are *popsize* chromosomes, and evaluate all chromosomes. The evaluation of a chromosome is a process to synthesize the chromosome according to a rule into a schedule with an objective value. Actually, this “Initial generation” step is the same procedure as the random search in Section 5.

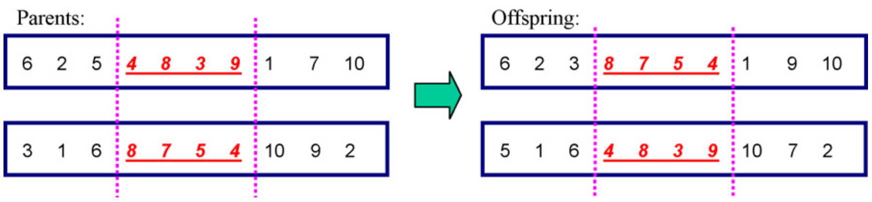

Fig. 8. Illustration of the process of PMX.

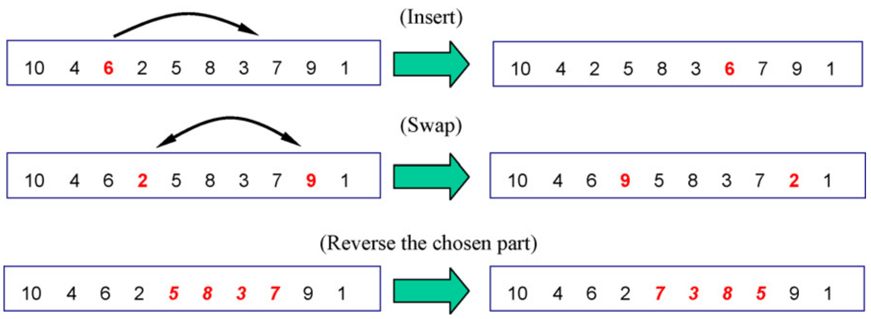

Fig. 9. Illustration of the process of mutation.

Table 8
Results of 10-order/4-unit instance in Example 1 by GA (min C_{max})

Method	Best	Mean C_{max}	Mean CPU time (s)	Mean iterations	Rule quality
GA_R1 (FAU)[a]	17.35	17.73	0.05	19.4	Good
GA_R2 (SCP)	17.35	18.78	0.05	20.2	Poor
GA_R3 (SPT)	26.60	26.60	<0.05	2	Poor
GA_R4 (EST)	17.35	17.75	0.05	19	Good
GA_R5 (SPsPT)	**17.35**	**17.35**	<0.05	12.8	Good
GA_R6 (SCPT)	18.30	18.30	<0.05	13	Poor
GA_R7(ECT)	**17.35**	**17.35**	<0.05	14.2	Good

[a] GA_Rn stands for GA combined with Rule *n*.

- *Step 2*. Selection: Select *xsize* better chromosomes in each generation to crossover, and *msize* rest chromosomes to mutate.
- *Step 3*. Crossover: Conduct PMX crossover to the *xsize* better chromosomes, and generate *xsize* new chromosomes.
- *Step 4*. Mutation: Conduct reversion mutation to the *msize* rest chromosomes, and generate *msize* new chromosomes.
- *Step 5*. Evaluation and new generation: The offspring (consist of the *xsize* new chromosomes and the *msize* new chromosomes) are evaluated. Let the parents and the offspring queue according to their objective values, and select the former *popsize* best chromosomes as the new generation.
- *Step 6*. Test the termination condition: If the termination condition is met, then go to Step 7; else go to Step 2 for continuing iteration.
- *Step 7*. Output the results: Output the best chromosome, its objective value and the data for Gantt chart, this is the best solution found by GA.

6.3. Computational results by GA with different rules

GA has been combined with the seven rules respectively for solving the small sample instance. Table 8 presents the results. For each method, 10 tests of computation were performed. The makespan of the best schedules are 17.35. It can be observed that from Table 8:

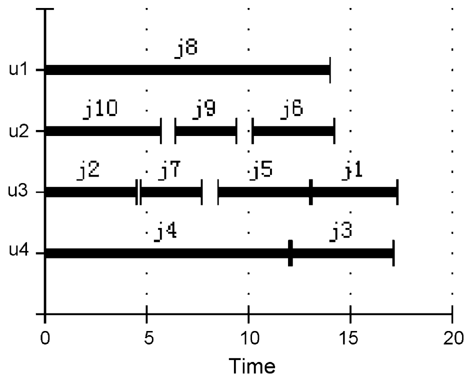

Fig. 11. A schedule by GA_R7 ($C_{max} = 17.35$).

(1) *Rule 5* and *Rule 7* are the best rules that enable the GA to obtain the shortest makespan in every test. (2) *Rule 1*, *Rule 2* and *Rule 4* are also good rules, but not enabling the GA to obtain the shortest makespan in every test. (3) *Rule 3 and Rule 6* are poor quality rules that never enable the GA to obtain the shortest makespan. *Rule 3* is the worst one. The best schedule obtained by GA combined with *Rule 7* is shown in Fig. 11, where $\pi = (2, 8, 10, 4, 7, 9, 5, 6, 3, 1)$, $f(\pi) = C_{max} = 17.35$.

Table 9
Differences between Examples 1–3

N, M	Example 1 $N = 50, M = 4$	Example 2 $N = 50, M = 4$	Example 3 $N = 200, M = 16$
c_{ij} and p_{iu}	Without CP constraints, no c_{ij} in the changeover time matrix is null and no p_{iu} in process time matrix is null.	With CP constraints, some of c_{ij} in the changeover time matrix are null and some of p_{iu} in process time matrix are null.	Without CP constraints, no c_{ij} in the changeover time matrix is null and no p_{iu} in process time matrix is null. c_{ij} and p_{iu} are generated randomly: $c_{ij} \in (0.10, 2.00)$, $p_{iu} \in (5.0, 20.00)$.
d_i	With a finite value	With a finite value	d_i is generated by random, $d_j \in [20,60]$
or_i	$\mathrm{or}_i = 0$	With a finite value	or_i is generated by random, $\mathrm{or}_j \in [0,6]$
ur_u	$\mathrm{ur}_u = 0$	With a finite value	ur_u is generated by random, $\mathrm{ur}_u \in [0,3]$

Table 10
Results of different size instances in Example 1 by MILPs (min C_{max})

N	Basic model				Revised model (with pre-ordering heuristic)			
	100000 iter.		10000000 iter.		100000 iter.		1000000 iter.	
	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}
8[a]	1.45	14.00	1.45	14.00	1.75	15.20	1.75	15.20
10	19.28	20.25	275.08	19.50	18.93	18.20	190.48	18.20
15	29.80	31.75	333.28	29.82	27.77	27.75	314.56	27.05
20	45.35	42.05	482.44	42.05	41.91	35.45	436.68	35.45
50	325.77	111.55	3191.87	105.50	236.70	97.00	2404.13	97.00

[a] Optimality was achieved in short CPU time.

7. Case study

The following five examples are used for case study. Example 1 and Example 2 were widely studied in the literature (Cerda et al., 1997; Hui & Gupta, 2001; Karimi & McDonald, 1997; and Chen et al., 2002). The authors often solved small-size problems (less than 20 orders) by using MILP. In this paper, the problems are extended up to 50 orders, and it is very difficult for MILP to obtain acceptable solutions to these large-size problems. Example 3 is a randomly generated problem, a very large general problem. The changeovers in Examples 1–3 are sequence-dependant, but unit independent. Example 4 comes from the work by Castro and Grossmann (2006), in which no changeover time exists. Example 5 comes from the work by Castro et al. (2006), in which the changeover times are not only sequence-dependant, but also unit-dependant. The data for Examples 1–3 are presented in Supplementary data (Appendix A). Table 9 shows the differences between the Examples 1–3.

In Examples 1 and 3, there is no CP constraint. But in Example 2, there are CP constraints. In order to increase feasibility of the solutions explored and reduce the search time, a penalty method has been adopted to change the problem with CP constraints into one without CP constraint.

To compare the performance of MILP, RS and GA, these three approaches are used to solve different size instances in Example 1. For the large-size problems, e.g. the 50-order onward instances in Example 3, since it is obvious that MILP cannot provide good enough solutions for comparison, the results by MILP are not presented. Examples 4 and 5 are recently studied SMSP, for which the efficiency of the newly proposed MILP models increase greatly, but the MILP models still show their limitation in solving large-size problems. Example 4 involves 30 orders without any changeovers, and Example 5 involves 20 orders with both sequence- and unit-dependant changeovers that greatly increase the complexity MILP models. However, the rule-based GA shows better performance then the efficiency-improved MILP models for Example 5.

7.1. *Example 1*

In this section, we first solve Example 1 using the basic MILP model and the revised MILP model developed by Hui and Gupta (2001) and then solve Example 1 using our proposed RS and GA. When using GAMS and OSL for solving the MILP models, iterations is limited to 100,000 or 1,000,000 for the computation. For the small-size instances, if an optimal solution is obtained, the computational process will stop with a small number of iterations. Table 10 shows the results of Example 1 by MILPs. From the results in Table 10, it can be noticed that: (1) Among the five instances, only the 8-order instance is solved to optimality. (2) As the problem size increases linearly, from 8-order/4-unit to 50-order/4-unit, the computational time of MILP model increases greatly, from 2 s to about 300 s (with a termination criterion of 100,000 iterations), or from 2 s to about 3000 s (with a termination criterion of 1,000,000 iterations). In fact, even the iterations increase up to 10,000,000 for the 50-order instance, which need about 17 h computational time, the solution obtained is still far from the best known solution. (3) Although the solutions by the revised MILP model are somewhat better, the CPU times do not reduce greatly. (4) Although the iterations increase by 10 times, the objective values by the basic model decrease very little, and the objective values by the revised model almost have no change.

Tables 11 and 12 show the results by our proposed RS and GA, respectively. Analyzing the results, the following observations can be noticed: (1) Both RS and GA obtain the optimum for the 8-order instance within very short CPU time. (2) For the other four instances, the solutions by RS and GA are optimal or near-optimal, though not able to be proved by the methods themselves. (3) The Rules behave differently. *Rule 5* and *Rule 7* show good performance in solving the three small-size instances, whereas *Rule 6* performs better for the two large-size instances.

Table 13 presents the performance comparison of MILP, RS and GA for Example 1. With the problem size increasing, the performance difference between MILP and RS or GA becomes more obvious, which can be seen from the solution quality and the CPU time. Fig. 12 shows the performance difference clearly.

Figs. 13–15 show the schedules of the 50-order instances by MILP, RS and GA, respectively. The final schedule obtained by MILP has evident and big time intervals; however, the final schedules obtained by RS and GA are even, uniform and acceptable.

Table 11
Results of different size instances in Example 1 by RS (min C_{max})

N/M	*popsize*	RS_R5		RS_R6		RS_R7	
		CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}
8/4	200	<0.055	**14.00**	<0.055	15.25	<0.055	**14.00**
10/4	300	<0.055	**17.35**	<0.055	18.30	<0.055	**17.35**
15/4	400	<0.055	**22.60**	<0.055	23.87	<0.055	**22.64**
20/4	500	0.055	28.70	0.055	**28.54**	0.055	28.60
50/4	1000	0.11	77.30	0.11	**73.50**	0.11	76.65

Table 12
Results of different size instances in Example 1 by GA (min C_{max})

N/M	*popsize*	GA_R5		GA_R6		GA_R7	
		CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}
8/4	200	0.055	**14.00**	0.055	15.25	0.055	**14.00**
10/4	200	0.055	**17.35**	0.055	18.30	0.055	**17.35**
15/4	400	0.22	**22.05**	0.22	23.35	0.27	**22.05**
20/4	400	0.66	27.94	0.49	**27.45**	0.49	27.95
50/4	600	4.07	73.03	2.97	**70.30**	3.68	72.75

Table 13
Comparison of MILP, RS and GA for Example 1 (min C_{max})

N/M	MILP		RS_R7		GA_R7	
	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}
8/4*	**1.45**	**14.00**	<0.055	**14.00**	0.055	**14.00**
10/4	190.48	18.20	<0.055	**17.35**	0.055	**17.35**
15/4	314.56	27.05	<0.055	**22.64**	0.27	**22.05**
20/4	436.68	35.45	0.055	28.60	0.49	27.95
50/4	2404.13	**97.00**	0.11	76.65	3.68	72.75

* Optimality was achieved in relatively short CPU time.

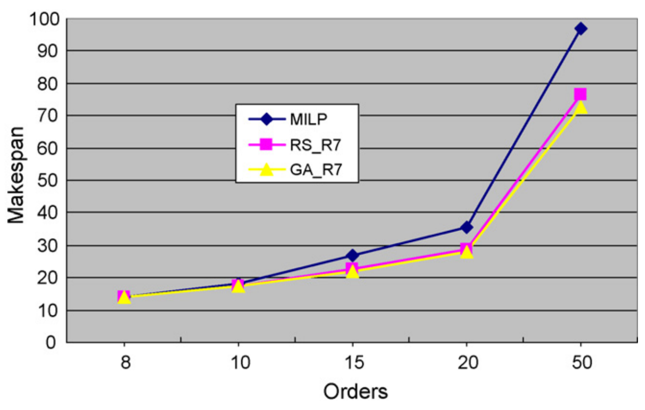

Fig. 12. Performance contrast of MILP, RS and GA for Example 1 (min C_{max}).

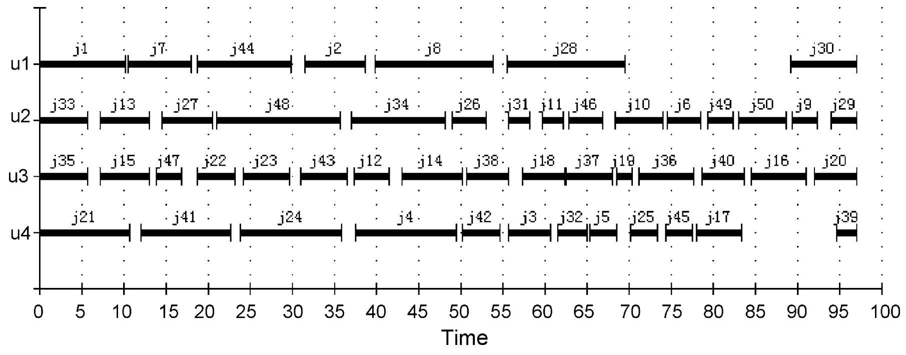

Fig. 13. A schedule of the 50-order instance in Example 1 by MILP (C_{max} = 97.00).

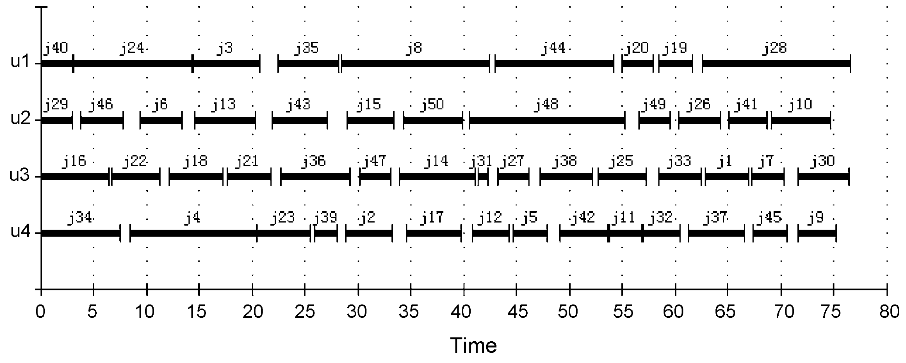

Fig. 14. A schedule of the 50-order instance in Example 1 by RS_R7 (C_{max} = 76.65).

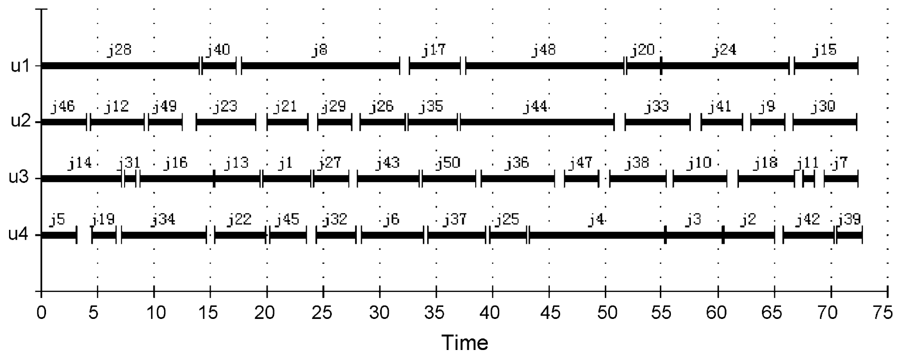

Fig. 15. A schedule of the 50-order instance in Example 1 by GA_R7 (C_{max} = 72.75).

To compare the performance of the seven rules in RS and GA, the 50-order/4-unit instance of Example 1 was solved by RS/GA combined with different rules. For each method, 10 tests were conducted. Table 14 and Table 15 present the results. From Tables 14 and 15, it can be seen that: (1) In both RS and GA, *Rules 5-7* show better performance than the other four rules, *Rule 6* is the best one which enables the GA to obtain the shortest makespan 70.30 among the seven rules. (2) With the problem condition (e.g. problem size) changed, the effective rule to this instance changes to be *Rule 6*.

Table 16 presents the results of tardiness minimization by MILP, RS and GA. All the three methods can obtain the optimality for the small-size instances (with 8 or 10 orders). However, for the large-size instances, GA shows its great advantage.

Table 14
Results of the 50-order/4-unit instance of Example 1 by RS (min C_{max})

Method	Best	Mean C_{max}	Mean CPU time (s)	Rule quality
RS_R1 (FAU)	85.82	90.39	0.11	Poor
RS_R2 (SCP)	86.00	91.55	0.11	Poor
RS_R3 (SPT)	98.95	99.6	0.05	Poor
RS_R4 (EST)	86.92	89.22	0.11	Poor
RS_R5 (SPsPT)	**77.30**	**79.09**	0.11	Good
RS_R6 (SCPT)	**73.50**	**75.12**	0.11	Good
RS_R7 (ECT)	**76.65**	**78.28**	0.11	Good

Table 15
Results of the 50-order/4-unit instance of Example 1 by GA (min C_{max})

Method	Best	Mean C_{max}	Mean CPU time (s)	Mean iterations
GA_R1 (FAU)	78.70	81.22	3.16	60.4
GA_R2 (SCT)	80.54	83.64	3.14	60.2
GA_R3 (SPT)	95.40	96.18	1.95	43.7
GA_R4 (EST)	79.30	81.48	3.37	63.7
GA_R5 (SPsPT)	**73.03**	74.15	3.48	67.8
GA_R6 (SCPT)	**70.30**	71.03	3.06	63.4
GA_R7 (ECT)	**72.75**	73.88	3.24	64.2

Table 16
Results of different size instances in Example 1 by MILP, RS and GA (min T)

N/*M*	Revisd MILP			RS_R7			GA_R7		
	Iterations	CPU(s)	T	*popsize*	CPU(s)	T	*popsize*	CPU	T
8/4	368	0.08	0.00	200	<0.055	0.00	200	0.055	0.00
10/4	199	0.05	0.00	300	<0.055	0.00	200	0.055	0.00
15/4	1000000	261.29	0.12	400	<0.055	0.00	400	0.11	0.00
20/4	1000000	410.38	14.40	500	0.055	0.00	400	0.16	0.00
50/4[a]	1000000	2278.31	324.82	1000	0.11	148.91	600	4.07	0.35

[a] For the 50-order instance, the due dates of the orders in the Supporting Information are doubled.

Table 17
Problem data of the 10-order instance of Example 2 (with CP constraints)

	Changeover times (c_{ij})										d_i	or_i	Process times (p_{iu})			
	j1	j2	j3	j4	j5	j6	j7	j8	j9	j10			u1	u2	u3	u4
i1	–	–	–	–	–	0.65	–	–	0.85	0.40	10	0	10.20	–	–	–
i2	–	–	1.10	–	–	–	–	–	–	–	22	5	–	–	4.50	–
i3	1.00	0.15	–	–	–	–	0.30	–	1.60	0.20	25	0	6.25	–	5.50	–
i4	–	–	–	–	0.05	–	–	–	–	0.50	20	6	–	13.60	–	–
i5	–	–	–	0.30	–	0.70	0.90	0.60	–	–	28	0	–	8.40	–	3.40
i6	1.40	–	0.30	0.70	–	–	–	–	1.20	–	30	2	9.60	9.00	–	–
i7	–	1.80	–	–	0.85	–	–	0.45	–	–	17	3	–	–	3.15	6.60
i8	–	–	–	–	–	–	1.65	–	–	–	23	0	–	–	–	16.80
i9	2.10	–	1.25	–	–	0.80	–	–	–	0.65	30	2	4.80	–	–	–
i10	1.50	–	0.60	0.75	0.50	–	–	–	0.70	–	30	6	7.80	5.70	–	–
d_i: Due date; or_i: order release time; ur_u: unit release time												ur_u	0	3	2	3

Table 18
Results of different size instances in Example 2 by the basic MILP (min C_{max} and T)

N/*M*	Min C_{max}				Min T			
	100000 iter.		1000000 iter.		100000 iter.		1000000 iter.	
	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	T	CPU(s)	T
10/4	0.50	26.25	0.50	26.25	0.38	1.00	0.38	1.00
20/4	32.95	59.24	374.51	59.24	43.59	55.29	457.80	44.53
50/4[a]	149.74	137.65	1385.83	137.65	163.65	852.31	1764.65	659.20

[a] For the 50-order instance, the due dates of the orders in the Supporting Information are doubled.

7.2. *Example 2*

Example 2 is an SMSP with CP constraints. Table 17 presents the problem data of the 10-order instance. In this section, we first solve Example 2 using the basic MILP model developed by Hui and Gupta (2001) and then solved Example 2 using our proposed RS and GA. When using GAMS and OSL for solving the MILP models, iterations is limited to 100,000 or 1,000,000 for the computation. For the small-size instances, if an optimal solution is obtained, the computational process will stop with a small number of iterations. Table 18 shows the results of Example 2 by MILP.

In GA for solving SMSP with CP constraints, infeasible chromosomes will surely be generated in these steps: "Initial generation", "Crossover", "Mutation". In this case, new feasible chromosomes should be re-generated to replace the infeasible ones. It will take a lot of CPU time for GA to generate feasible chromosomes. To increase computing speed of GA for problems with CP constraints, a penalty method has been adopted (He & Hui, 2006), which gives the forbidden changeovers or forbidden processes a large penalty numerical value: a great changeover time or process time. For instance, change the null values ("-") in changeover time matrix and process time matrix into 100.0. Therefore, every arbitrary chromosome becomes a legal or feasible one. As a result, CPU time is greatly reduced. Moreover, GA obtained similar solutions as before.

To compare the performance of the seven rules, three instances in Example 2 are solved by RS or GA combined with different rules. For each method, 10 tests were conducted. Since *Rules 1-4* demonstrated poor performance in minimizing the makespan, the results by *Rules 1-4* are not presented here. Tables 19 and 20 present the results. It is clear that *Rule 7* is the best rule for makespan minimization of 50-order instance in Example 2. For the same size instances (50 orders over 4 units), the effective rule (*Rule 7*) for Example 2 is different from the rule (*Rule 6*) for Example 1. The main difference between these two examples is that Example 3 has CP constraints, but Example 2 has no CP constraint.

Table 19
Results of different size instances in Example 2 by RS (min C_{max})

N/*M*	*popsize*	RS_R5		RS_R6		RS_R7	
		CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}
10/4	300	<0.055	26.25	<0.055	26.25	<0.055	**26.25**
20/4	500	<0.055	39.80	<0.055	39.39	<0.055	**38.77**
50/4	1000	0.11	278.37	0.11	118.04	0.11	**109.70**

Table 20
Results of different size instances in Example 2 by GA (min C_{max})

N/M	*popsize*	GA_R5		GA_R6		GA_R7	
		CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}
10/4	200	<0.055	26.25	<0.055	26.25	<0.055	**26.25**
20/4	400	0.44	37.47	0.44	37.20	0.44	**37.20**
50/4	600	5.55	134.35	3.52	96.70	3.35	**95.75**

Table 21
Results of different size instances in Example 2 by RS and GA (min T)

N/M	RS_R7			GA_R7		
	popsize	CPU(s)	*T*	*popsize*	CPU(s)	*T*
10/4	300	0.055	0.00	200	0.055	0.00
20/4	500	0.055	0.00	400	0.11	0.00
50/4[a]	1000	0.11	696.93	600	4.12	397.14

[a] For the 50-order instance, the due dates of the orders in the Supporting Information are doubled.

Table 21 presents the results of different size instances in Example 2 by RS and GA for tardiness minimization. For both 10-order/4-unit and 20-order/4-unit instances, RS_R7and GA_R7 find the schedule with zero tardiness. For the 50-order/4-unit instance, GA_R7 obtained a much better schedule than MILP and RS_R7.

7.3. Example 3

Example 3 is a randomly generated problem that involves up to 200 orders to be processed by 16 units. Table 22 presents the problem data of the 16-order/3-unit instance in Example 3. In this section, GA combined with different rules are first used for makespan minimization of three small-size instances, 8-order, 10-order and 16-order over 3-unit. It is very easy for GA to obtain the best known solutions, $C_{max} = 29.29$ for the 8-order/3-unit instance, $C_{max} = 33.33$ for the 10-order/3-unit instance and $C_{max} = 52.92$ for the 16-order/3-unit instance. Table 23 presents the results of three small instances in Example 3 by GA combined with different rules.

And then six instances are solved by the MILP models proposed by Hui and Gupta (2001), and Table 24 shows the results. From Table 24, we can see that: (1) The optimum (29.29) for the 8-order/3-unit instance is achieved by both the basic model and the revised model. (2) The optimum (33.33) for the 10-order/3-unit instance was achieved by tuning the Big *M* in the basic model. For this instance, the revised model also achieved the optimum (33.84) to the model (not to the original problem). (3) From the 16-order/3-unit instance onward, the solutions is far away from the best known solutions obtained by GA later, though the search time gets up to 1,2591.38 s (about 3.5 h) for 50-order/5-unit instance.

Table 23
Results of three small instances in Example 3 by GA combined with different rules

Method	N/M					
	8/3		10/3		16/3	
	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}
GA_R1 (FAU)	0.055	29.29	0.055	33.33	0.27	54.61
GA_R2 (SCT)	0.055	**30.96**	0.11	33.33	0.27	56.00
GA_R3 (SPT)	0.055	**40.05**	0.055	**40.05**	0.11	63.32
GA_R4 (EST)	0.055	29.29	0.11	33.33	0.16	54.48
GA_R5 (SPsPT)	0.055	29.29	0.055	33.33	0.16	**52.92**
GA_R6 (SCPT)	0.055	29.29	0.055	**35.82**	0.11	56.25
GA_R7 (ECT)	0.055	29.29	0.055	33.33	0.22	**52.92**

Table 25 presents the results of different size instances in Example 3 by GA combined with *Rules 5, 6 and 7* for makespan minimization. GA_R6 obtain the best known solutions for all these large-size instances. Fig. 16 shows the schedule the 200-order/16-unit instance in Example 3 by GA_R6.

7.4. Example 4

Example 4 is taken from the recent work of Castro and Grossmann (2006), in which the objective is cost and earliness minimization. In our work, the objective is to minimize the makespan. For makespan minimization, the multiple-time-grid, continuous-time formulation by Castro and Grossmann (2006) is not the best efficient mathematical programming method. The constrain-programming method and the discrete-time formulation were found to be the best mathematical programming methods (Castro & Grossmann, 2005). However, these two methods have the disadvantage of considering integer data and show a rapid decrease in performance with an increase in problem complexity. The problem data for Example 4 is presented in Table 26. There is no changeover time in this example. Therefore, *Rule 1* is equivalent to *Rule 4*, *Rule 3* equivalent to *Rule 6*, and *Rule 5* equivalent to *Rule 7* (See Table 4). In this section, we still use the continuous-time MILP model (the basic one) by Hui and Gupta (2001) for solution, and then the proposed GA is applied to this example.

Table 22
Problem data of the 16-order/3-unit instance in Example 3

Changeover times (c_{ij})

	j1	j2	j3	j4	j5	j6	j7	j8	j9	j10	j11	j12	j13	j14	j15	j16
i1	–	1.67	0.13	1.60	0.18	0.78	1.78	1.94	1.87	0.25	1.38	1.66	1.21	0.68	0.76	0.60
i2	0.76	–	0.41	1.98	1.21	0.60	0.52	0.43	1.85	0.77	0.60	0.68	0.30	1.65	0.10	1.58
i3	0.78	1.67	–	1.47	0.67	0.94	1.62	0.27	0.43	1.68	1.78	0.66	0.28	0.99	0.25	1.35
i4	1.27	1.87	1.95	–	1.97	1.18	1.09	0.64	0.43	0.59	1.59	0.83	0.18	1.74	0.60	0.37
i5	1.67	1.60	1.74	0.60	–	1.88	1.59	0.76	0.56	1.73	1.62	0.99	0.27	1.32	1.44	0.98
i6	0.69	0.65	0.96	1.49	1.20	–	1.02	1.04	0.99	0.17	1.83	0.39	1.99	0.98	0.37	0.22
i7	1.91	1.79	1.03	0.63	1.93	0.74	–	1.64	1.50	1.29	1.92	1.69	0.58	1.24	0.44	1.03
i8	0.46	1.80	0.30	0.63	0.71	0.54	0.38	–	1.81	1.63	1.00	0.84	0.68	0.35	1.38	1.75
i9	0.55	0.77	1.34	1.48	1.24	1.80	0.51	1.24	–	1.27	1.25	0.95	1.86	1.66	0.54	0.19
i10	0.74	1.22	1.29	1.16	0.73	0.71	1.67	1.29	0.29	–	0.48	0.78	1.37	1.12	1.76	0.72
i11	0.48	0.75	1.58	1.16	0.33	1.09	1.80	1.62	1.90	0.39	–	1.52	1.06	0.82	1.54	1.70
i12	1.13	1.19	0.63	1.90	0.61	1.32	1.07	0.44	1.17	1.62	0.21	–	1.10	0.80	0.30	1.13
i13	1.52	1.68	0.20	1.16	1.02	1.35	1.73	0.68	1.12	0.73	1.92	1.03	–	1.63	0.44	1.67
i14	1.38	0.67	0.57	1.30	1.51	1.47	0.48	1.66	0.24	1.82	1.32	0.48	0.89	–	0.95	1.61
i15	1.15	0.97	1.59	0.59	0.60	0.34	0.15	1.46	0.15	1.70	1.92	1.21	0.98	1.94	–	0.29
i16	0.21	0.74	0.21	0.95	1.30	1.84	0.70	0.48	0.20	1.08	1.15	1.73	0.83	0.97	0.25	–

Process times (p_{iu}), due dates (d_i), order release times (or_i), unit release times (ur_u)

	i1	i2	i3	i4	i5	i6	i7	i8	i9	i10	i11	i12	i13	i14	i15	i16	ur_u
u1	15.26	9.00	10.02	19.84	5.60	10.62	10.03	6.45	5.53	6.67	8.56	14.84	9.76	8.74	19.04	15.35	3
u2	13.66	6.20	7.58	12.63	6.73	8.91	9.98	15.68	14.77	7.19	17.15	16.27	11.74	14.37	6.78	16.50	1
u3	12.54	8.80	18.15	18.20	10.00	8.70	19.21	11.69	17.36	13.98	6.53	17.85	19.24	11.09	13.50	11.92	2
d_i	31	23	25	49	39	42	26	45	55	29	40	22	56	44	43	41	
or_i	0	0	3	2	3	5	0	2	4	0	5	4	1	0	2	2	

Table 24
Results of different size instances in Example 3 by MILPs (min C_{max})

N/*M*	Basic model			Revised model (with pre-ordering heuristic)		
	Iterations	CPU(s)	C_{max}	Iterations	CPU(s)	C_{max}
8/3[a]	167236	558.3	29.29	19229	3.40	29.29
10/3	3000000	1325.56	34.18	190281	37.20	33.84
16/3	3000000	1337.41	61.57	3000000	1178.90	58.73
20/5	3000000	1687.15	50.54	3000000	1510.43	43.06
32/5	3000000	3777.71	83.98	3000000	3235.59	84.85
50/5	3000000	12591.38	132.73	3000000	9376.35	137.92

[a] Optimality was achieved in relatively short CPU time.

Table 25
Results of different size instances in Example 3 by GA (min C_{max})

N/*M*	*popsize*	GA_R5		GA_R6		GA_R7	
		CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}
20/5	300	0.38	34.06	0.33	33.98	0.38	34.63
32/5	300	0.99	57.27	0.93	54.76	0.93	57.11
50/5	300	2.47	90.32	1.10	85.68	1.59	89.85
80/5	400	7.75	141.33	3.52	133.70	4.23	140.98
100/8	400	6.81	102.71	7.31	96.58	7.03	100.12
200/16	400	33.23	97.57	31.63	92.34	97.27	30.55

Table 27 presents the results of different size instances in Example 4 by MILP and GA for makespan minimization, where due dates are still not considered as constraints as before. The termination criterion of the MILP model is set to be 3,000,000 iterations. In GA, *popsize* = 600 for all instances. It can be seen that MILP only find the optimum for the 8-order/3-unit instance, the solutions by MILP for the other instances are far from the best solutions by GA. For the instances under 15 orders, *Rule 1* (or *Rule 4*) and *Rule 5* (or *Rule 7*) show good performance in GA; but for the instances above 15 orders, *Rule 5* (or *Rule 7*) shows better performance than the other rules.

If the due dates are considered as constraints to be satisfied, the compound objective Eq. (4) can be applied, where if the total tardiness is equal to zero, the due dates are satisfied. Table 28 presents the results of different size instances in Example 4 by GA for the minimization of the compound objective TC. We can notice that, for the instances under 15 orders, both *Rule 1* (or *Rule 4)* and *Rule 5* (or *Rule 7*) show good performance in GA; but for the instances above 15 orders, *Rule 5* (or *Rule 7*) shows better performance than the other rules. Both *Rule 1* and *Rule 5* enable GA find solutions with zero tardiness.

7.5. Example 5

Example 5 is taken from the recent work of Castro et al. (2006), in which the objective is cost, makespan and earliness minimization. The problem data of the instances in this example can be found in the Supporting Information to the work by Castro et al. (2006), or can be provided by the corresponding author, Castro. The most distinction of Example 5 from Example 4 and the other previous examples is that there are sequence- and unit-dependant changeover times between products, which makes it more difficult or complex for MILP methods to solve the problems. Castro et al. (2006) proposed two new multiple-time-grid, continuous-time MILP models, CT4I and CT3I, the former used four-index binary variables to handle sequence-and unit-dependant changeovers, and the later used three-index binary variables for this purpose. In this section, only makespan minimization is performed.

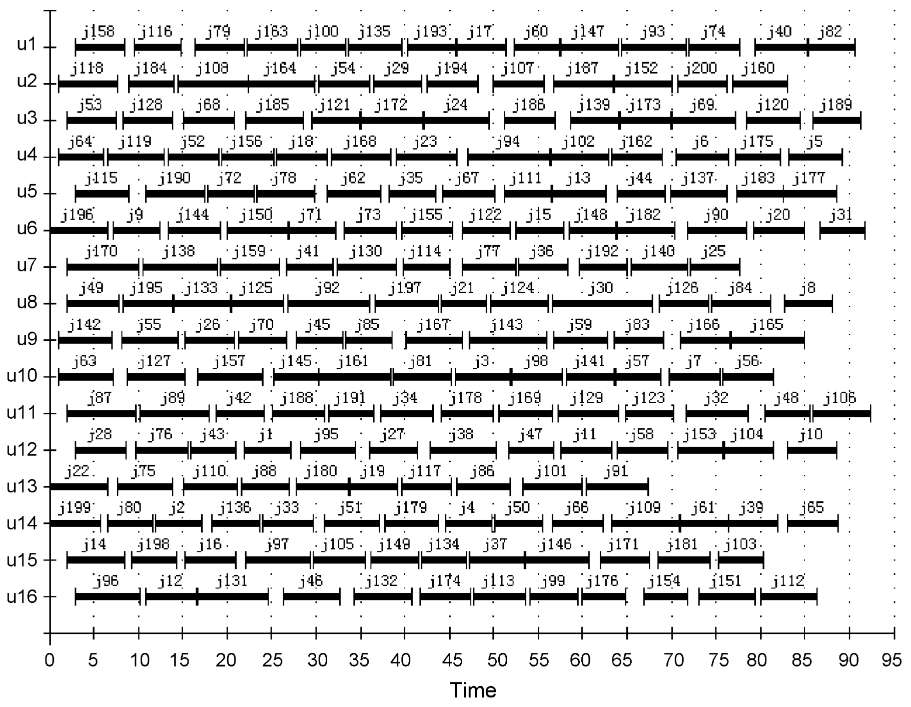

Fig. 16. A schedule of the 200-order/16-unit instance in Example 3 by GA_R6 (C_{max} = 92.34).

Table 26
Problem data for Example 4

		p_{iu}					
Order	or_i	d_i	u1	u2	u3	u4	u5
i1	66	110	30	53	39	36	56
i2	40	188	30	50	50	57	43
i3	65	163	35	45	42	45	41
i4	28	137	46	40	38	41	31
i5	36	221	46	34	50	55	57
i6	10	126	47	49	31	29	54
i7	56	286	38	55	33	34	51
i8	50	237	29	57	42	42	49
i9	20	254	48	54	50	38	40
i10	26	119	32	39	50	42	39
i11	46	229	33	48	49	53	34
i12	78	189	31	57	49	52	42
i13	88	159	53	40	42	44	36
i14	53	219	28	55	29	28	57
i15	46	281	51	58	33	53	40
i16	95	269	43	57	32	39	44
i17	94	200	38	39	45	37	49
i18	12	258	55	34	58	56	40
i19	72	142	54	53	49	44	38
i20	12	184	28	57	38	43	51
i21	66	294	33	55	36	43	48
i22	29	184	54	58	49	47	31
i23	2	295	48	54	49	33	31
i24	99	156	50	29	37	40	45
i25	81	142	43	53	41	33	38
i26	3	270	42	50	33	52	37
i27	45	277	41	54	57	43	49
i28	2	134	49	50	40	37	45
i29	16	170	54	37	48	48	43
i30	75	157	57	43	57	52	37

According to the comparative study of Castro et al. (2006), the constrain-programming (CP) model was the best overall MP method for makespan minimization because it was the fastest for P1M-P3M and could also find the best solution for P5M, for which the optimal solution was still unknown. Note, however, that its performance for P6M is rather weak, because the best solution found after more than 15 h of computational time, 237, is still far from the best known solution of 164 (found by discrete-time). It is clear from Table 29 that with the number of orders and equipment units increasing, the computational effort of MP models increases greatly.

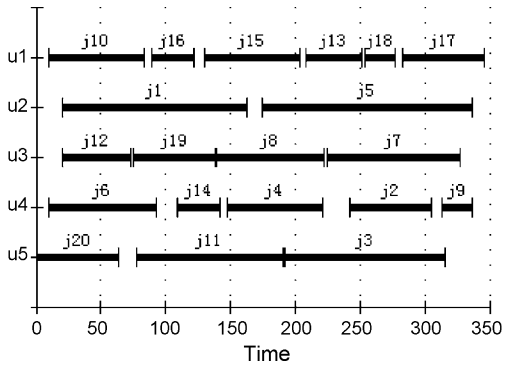

Fig. 17. A schedule of P5M in Example 5 by GA_R7 ($C_{max} = 346$).

The results of different size instances in Example 5 by GA_R7 for makespan minimization is also presented in Table 29, where *popsize* = 600 for all instances. To satisfy the due dates, the compound objective Eq. (4) is still used for this example. For all the instances, GA_R7 has obtained final solutions with zero tardiness. The CPU times for GA_R7 to find the optimal solutions to P1M and P2M are at the same level as CP. For P3M and P4M, GA_R7 spend far less CPU times to achieve the optimality than the MP models. The solutions for P5M and P6M by GA_R7 are very near to the best known solutions, but the CPU times of GA_R7 are far less than those of the MP models. Figs. 17 and 18 show the schedules by GA_R7 for P5M and P6M, respectively.

We should also notice that the change of changeovers from sequence-dependant to both sequence- and unit-dependant makes the MILP models much more complex, but this change has no negative affect to the performance of GA. That is to say, this change does not increase the difficulty of the programming of GA, and not reduce the solution speed of GA.

Table 27
Results of different size instances in Example 4 by MILP and GA (min C_{max})

	MILP (3000000 iter.)		GA_R1 (R4)		GA_R3 (R6)		GA_R5 (R7)	
N/M	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}
8/3	86.73	129[a]	0.05	129	0.05	164	0.11	129
10/3	1008.77	158	0.11	146	0.05	224	0.16	147
12/3	1079.13	191	0.16	178	0.05	288	0.16	178
15/5	1213.16	173	0.33	139	0.05	274	0.27	139
20/5	1605.62	209	0.71	170	0.11	288	0.60	160
25/5	2405.57	273	1.37	197	0.16	321	0.99	191
30/5	3311.02	298	1.21	238	0.22	362	1.98	222

[a] Optimum.

Table 28
Results of different size instances in Example 4 by GA (min *TC*)

	GA_R1 (R4)		GA_R3 (R6)		GA_R5 (R7)	
N/M	CPU(s)	C_{max}/T	CPU(s)	C_{max}/T	CPU(s)	C_{max}/T
8/3	0.11	**129/0**	0.05	164/25	0.05	129/0
10/3	0.16	**151/0**	0.05	224/112	0.16	151/0
12/3	0.22	**180/0**	0.05	288/316	0.22	180/0
15/5	0.44	**139/0**	0.05	274/222	0.27	139/0
20/5	0.99	173/0	0.11	288/388	0.44	**162/0**
25/5	1.21	211/0	0.16	321/537	1.21	**191/0**
30/5	1.32	243/0	0.22	362/729	1.76	**226/0**

Table 29
Comparison of computational performance (CPUs) of MILP and GA (min C_{max})

Problem (*N*/*M*)	Optimum	CT4I	CT3I	CP	GA_R7
P1M (12/3)	409	15.5	193	0.98	0.27
P2M (12/3)	171	3.55	11.7	0.84	0.27
P3M (15/5)	291	201	8373	122	0.38
P4M (15/5)	147	1435	1.76	702	0.33
P5M (20/5)	337?	8782 (338)	36692 (347)	54000 (337)	0.77 (346)
P6M (20/5)	164?	6290 (168)	20000 (166)	55000 (237)	0.49 (167)

8. Discussion

8.1. Other objectives

The objective in this work is makespan or tardiness minimization. From another our work (He & Hui, 2007a), it can be known that to synthesize an order sequence (a chromosome in GA) into a schedule, forward assignment or backward assignment strategy can be applied according different objectives. In fact, forward assignment method is adopted for makespan or tardiness minimization in this work. For earliness minimization, backward assignment strategy should be used. Another point is that the rules summarized in Table 4 are suitable for time-based makespan or tardiness minimization, not for cost-based objectives. That is why we do not present the results of cost or earliness minimization in Examples 4 and 5.

8.2. Rule selection and rule combination

For different scheduling objectives, like makespan, total tardiness, total earliness and cost minimization, there are different heuristic rules to be employed. The effectiveness of a heuristic rule depends on the scheduling objective and the prevailing shop or plant conditions. In practice, in order to select a suitable rule from a candidate rule base, considerable simulation experiments have to be conducted. So we wonder, whether rules can be chosen automatically by the algorithm itself, according to different scheduling objectives, problem type and the prevailing shop or plant conditions, or not? The answer is positive. We have developed a novel genetic algorithm-based method that evolves scheduling rules while evolving the solutions (He & Hui, 2006). Throughout the genetic evolution, a suitable heuristic rule or rule sequence will be automatically selected. Furthermore, in our work (He & Hui, 2007b), an automatic rule combination method has been proposed, which is more effective and efficient to solve large-size single/multistage multi-product scheduling problems.

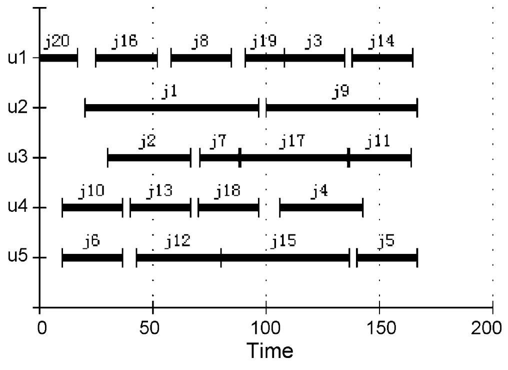

Fig. 18. A schedule of P6M in Example 5 by GA_R7 (C_{max} = 167).

9. Conclusions

This paper presents a heuristic rule-based genetic algorithm for single-stage multi-product scheduling problems (SMSP) in batch plants with parallel units, which, although cannot prove the optimality, is able to effectively and efficiently to solve the studied problems, especially suitable to solve large-size problems. In the proposed method, an order sequence is encoded into a chromosome in the genetic algorithm, which belongs to integer coding. In decoding chromosomes, through simulation experiments to a comprehensive set of heuristic rules, one of them is selected to synthesize the chromosomes into better schedules. Because of the evolutionary mechanism of the genetic algorithm, an initial generation of random chromosomes are evolved through computational iterations. At the end of computation, near or near-optimal solutions can be found. The proposed method can easily handle the forbidden changeovers and processes, and the both sequence- and unit-dependant changeovers that greatly increase difficulty or complexity of the MILP methods. The comparative study shows that it is very easy for the proposed method to find the optimal solutions to the small-size problems as the MILP models do; furthermore, this method can achieve the near-optimal solutions to large-size problems that are still very difficult for MILP models to solve.

Acknowledgement

The authors gratefully acknowledge financial support from Hong Kong RGC grant (No. 614005) and DAG05/06.EG23.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.compchemeng.2008.04.008.

References

Castro, P. M., Barbosa-Povoa, A. P., Matos, H. A., & Novais, A. Q. (2004). Simple continuous-time formulation for short-term scheduling of batch and continuous processes. *Industrial & Engineering Chemistry Research*, *43*, 105–118.

Castro, P. M., Barbosa-Povoa, A. P., & Novais, A. Q. (2005). Simultaneous design and scheduling of multipurpose plants using RTN-based continuous-time formulations. *Industrial & Engineering Chemistry Research*, *44*, 343–357.

Castro, P. M., & Grossmann, I. E. (2005). New continuous-time MILP model for the short-term scheduling of multistage batch plants. *Industrial & Engineering Chemistry Research*, *44*, 9175–9190.

Castro, P. M., & Grossmann, I. E. (2006). An efficient MILP model for the short-term scheduling of single stage batch plants. *Computers & Chemical Engineering*, *30*, 1003–1018.

Castro, P. M., Grossmann, I. E., & Novais, A. Q. (2006). Two new continuous-time models for the scheduling of multistage batch plants with sequence dependent changeovers. *Industrial & Engineering Chemistry Research*, *45*, 6210–6226.

Cerda, J., Henning, P., & Grossmann, I. E. (1997). A mixed integer linear programming model for short-term scheduling of single-stage multiproduct batch plants with parallel lines. *Industrial & Engineering Chemistry Research*, *36*, 1695–1707.

Chen, C., Liu, C., Feng, X., & Shao, H. (2002). Optimal short-term scheduling of multiproduct single-stage batch plants with parallel lines. *Industrial & Engineering Chemistry Research*, *41*, 1249–1260.

Chen, C., Neppalli, R. V., & Aljaber, N. (1996). Genetic algorithms applied to the continuous flow shop problem. *Computers & Industrial Engineering*, *30*, 919–929.

Das, H., Cummings, P. T., & Le Van, M. D. (1991). Scheduling of serial multiproduct batch processes via simulated annealing. *Computers & Chemical Engineering*, *14*, 1351–1362.

Floudas, C. A., & Lin, X. (2004). Continuous-time versus discrete-time approaches for scheduling of chemical processes: a review. *Computers & Chemical Engineering*, *28*, 2109–2129.

Giannelos, N. F., & Georgiadis, M. C. (2002). A simple new continuous-time formulation for short-term scheduling of multipurpose batch processes. *Industrial and Engineering Chemistry Research*, *41*, 2178–2184.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimisation and machine learning*. Addison-Wesley (Reading, Mass).

Goldberg, D. E., & Deb, K. (1991). In G. Rawlins (Ed.), *A comparative analysis of selection schemes used in genetic algorithms*. Foundations of genetic algorithms, Morgan Kaufmann.

He, Y., & Hui, C. W. (2006). Rule-evolutionary approach for single-stage multi-product scheduling with parallel units. *Industrial & Engineering Chemistry Research*, *45*, 4679–4692.

He, Y., & Hui, C. W. (2007a). Genetic algorithm for large-size multi-stage batch plant scheduling. *Chemical Engineering Science*, *62*, 1504–1527.

He, Y., & Hui, C. W. (2007b). Automatic rule combination approach for single-stage process scheduling problems. *AIChE Journal*, *53*, 2026–2047.

Hui, C. W., & Gupta, A. (2001). A bi-index continuous time MILP model for Short-term scheduling of single-stage multi-product batch plants with parallel line. *Industrial & Engineering Chemistry Research*, *40*, 5960–5967.

Hui, C. W., Gupta, A., & Meulen, H. (2000). A novel MILP formulation for short term scheduling of multistage multi-products batch plants with sequence-dependent constraints. *Computers & Chemical Engineering*, *24*, 1611–1617.

Jain, V., & Grossmann, I. E. (2001). Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing*, *13*(4), 258.

Jou, C. (2005). A genetic algorithm with sub-indexed partitioning genes and its application to production scheduling of parallel machines. *Computers & Industrial Engineering*, *48*, 39–54.

Karimi, I. A., & McDonald, C. M. (1997). Planning and scheduling of parallel semi-continuous process. 2. Short-term scheduling. *Industrial & Engineering Chemistry Research*, *36*, 2701–2714.

Kim, M. S., Jang, J. H., & Lee, I. B. (1996). Intelligent scheduling and monitoring for multi-product networked batch processes. *Computers & Chemical Engineering*, *20*, 1149–1154.

Kondili, E., Pantelides, C. C., & Sargent, R. W. H. (1993). A general algorithm for short-term scheduling of batch operations. Part 1. MELP formulation. *Computers & Chemical Engineering*, *17*, 211–227.

Ku, H. M., & Karimi, I. A. (1991). An Evaluation of Simulated Annealing for Batch Process Scheduling. *Industrial & Engineering Chemistry Research*, *30*, 163–169.

Ku, H. M., Rajagopalan, D., & Karimi, I. (1987). Scheduling in batch process. *Chemical Engineering Progress*, *83*, 25–34.

Lee, D. S., Vassiliadis, V. S., & Park, J. M. (2002). List-based threshhold-accepting algorithm for zero-wait scheduling of multiproduct batch plants. *Industrial & Engineering Chemistry Research*, *42*, 6579–6588.

Maravelias, C. T., & Grossmann, I. E. (2003a). New general continuous-time state-task network formulation for short-term scheduling of multipurpose batch plants. *Industrial & Engineering Chemistry Research*, *42*, 3056–3074.

Maravelias, C. T., & Grossmann, I. E. (2003b). Minimization of the makespan with a discrete-time state-task network Formulation. *Industrial and Engineering Chemistry Research*, *42*, 6252–6257.

Maravelias, C. T., & Grossmann, I. E. (2004). A hybrid MILP/CP decomposition approach for the continuous time scheduling of multipurpose batch plants. *Computers & Chemical Engineering*, *28*, 1921–1949.

Mendez, C. A., & Cerda, J. (2000). Optimal scheduling of a resource-constrained multiproduct batch plant supplying intermediates to nearby end-product facilities. *Computers & Chemical Engineering*, *24*, 369–376.

Mendez, C. A., Cerda, J., Grossmann, I. E., Harjunkoski, I., & Fahl, M. (2006). State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Computers & Chemical Engineering*, *30*, 913–946.

Mendez, C. A., Henning, G. P., & Cerda, J. (2000). Optimal scheduling of batch plants satisfying multiple product orders with different due-dates. *Computers & Chemical Engineering*, *24*, 2223–2245.

Mockus, L., & Reklaitis, G. V. (1996). Continuous time presentation in batch/semicontinuous process scheduling: randomized heuristic approach. *Computers & Chemical Engineering*, *20*, 1173–1177.

Pantelides, C. C. (1993). Unified frameworks for optimal process planning and scheduling. In D. W. T. Rippin, J. C. Hale, & J. Davis (Eds.), *Proceedings of the second international conference on foundations of computer-aided process operations, 253–274*. Colorado: Crested Butte.

Pinedo, M. (1995). *Scheduling Theory Algorithms and Systems*. Englewood Cliffs, NJ: Prentice Hall.

Pinto, J. M., & Grossmann, I. E. (1995). A continuous time mixed integer linear programming model for short term scheduling of multistage batch plants. *Industrial & Engineering Chemistry Research*, *34*, 3037–3051.

Pinto, J. M., & Grossmann, I. E. (1996). A continuous time MILP model for short term scheduling of batch plants with pre-ordering constraints. *Computers & Chemical Engineering*, *20*, P1197–1202.

Pinto, J. M., & Grossmann, I. E. (1998). Assignment and sequencing models for the scheduling of process systems. *Annals of Operations Research*, *81*, 433–466.

Pinto, J. M., Turkay, A., Bolio, B., & Grossmann, I. E. (1998). STBS: a continuous-time MILP optimization for short-term scheduling of batch plants. *Computers & Chemical Engineering*, *22*, 1297–11308.

Poon, P. W., & Carter, J. N. (1995). Genetic algorithm crossover operators for ordering applications. *Computers & Operations Research*, *22*, 135–147.

Reklaitis, G. V. (1982). Review of scheduling of process operations. In *AIChE Symposium Series, vol. 78* (pp. 119–133).

Reklaitis, G. V., & Mokus, L. (1995). Mathematical programming formulation for scheduling of batch operations based on non uniform time discretization. *Acta Chimica Slovenica*, *42*, 81–86.

Sundaramoorthy, A., & Karimi, I. A. (2005). A simpler better slotbased continuous-time formulation for short-term scheduling in multiproduct batch plants. *Chemical Engineering Science*, *60*, 2679–2702.