Department of Social Systems and Management

Discussion Paper Series

No. 1118

A genetic algorithm with MGG and demand crossover to solve dynamic flexible scheduling problem

Tunglun Tsai, Ryo Sato, Takao Terano

May 2005

UNIVERSITY OF TSUKUBA

Tsukuba, Ibaraki 305-8573

JAPAN

A genetic algorithm with MGG and demand crossover to solve dynamic flexible scheduling problem

Tunglun Tsai[*], Ryo Sato[**], Takao Terano[***]

[*]Faculty of Economics, Kanto Gakuen University

[**]Institute of Policy and Planning Sciences, University of Tsukuba

[***]Dept. of Computational Intelligence and Systems Science, Tokyo Institute of Technology

Abstract

A genetic algorithm, called MDGA, is proposed for practical scheduling, where bills of materials of parts, routings of production operations, and work-in-process inventories on hand and in near future, are taken into consideration. The scheduling problem is called a dynamic flexible scheduling (DFS) problem. The MDGA algorithm uses the concept of basket of requirements in representation of chromosome. MDGA reproduces a population of chromosomes with the principle of minimum generation gap (Yamamura et al., 1996) instead of simple tournament selection in usual genetic algorithm.

In order to demonstrate the correctness of MDGA, a comparison with exhaustive search is provided, which also shows the difficulty in solving the DFS problem. By applying MDGA to a usual job shop scheduling problem, which is a simplified DFS problem, the effectiveness of MDGA is shown to be satisfactory. Finally, since MDGA has many parameters, it is examined how they effect on solution-search process.

***Keywords*:** Genetic algorithm; Agile production planning and control; Work-in-process; Demand crossover; Minimal generation gap

---

[**]Corresponding author: Ryo Sato
Phone (office): +81-298-53-5543 Facsimile: +81-298-55-3849
E-Mail: rsato@sk.tsukuba.ac.jp

1. Introduction

Nobody knows what will happen in the future. However, something must be predicted before it becomes clear due to the leadtime. In the competitive market, for example, it is almost impossible to begin a production after the actual demand is known. Based on information of the present, decision makers always forecast the future.

Hopp and Spearman (2000) pointed out three laws of forecasting: (1) forecasts are always wrong, (2) detailed forecasts are worse than aggregate forecasts, and (3) the further into the future, the less reliable the forecast will be. The second law explains the reason why production planning begins with master production schedule (MPS), which plans the long-term requirements of the product family. Subsequently, material requirements planning (MRP) is used to plans the short-term requirements of an individual product. The third law reveals that the less reliable forecast should be revised by some new information.

In the field of production management, researches try to build a model to predict the future demand. The first law does not disparage the activity of forecasting, but call attention to the importance of forecast revision. Sato and Tsai (2004) proposed agile production planning and control system (APPCS) to incorporate a change into production system and provided a methodology to respond to the change agilely and simultaneously. Once there is a notification of change, APPCS generates another feasible schedule based on work-in-process (WIP). Tsai and Sato (2004) gave a UML model of APPCS to show the realizability of APPCS. The schedule developed by APPCS is both practical and feasible, because it is compatible with the product data that has the same structure detail with a commercially available enterprise resource planning (ERP) package and an advanced planning and scheduling (APS) system.

APPCS provides a feasible schedule, but the schedule is not necessarily good. For a plant, a good schedule is the schedule that achieves its own goal and reflects requirements of the

market. The goal varies among problems and researches. For most of the scheduling problems, it is difficult to meet all the goals. There may be conflict among different goals. Kacem et al. (2002) proposed an approach to minimize makespan and total processing time (workload) for a flexible job shop schedule problem. The problem is different from the general job shop scheduling problem because it assumes the performance of the machines in a work center is different. Assigning a fast machine to an operation minimizes both makespan and workload at first. However as the capacity of the fast machines approaches to full, the optimization faces a dilemma of continuously choosing a fast machine to increase makespan, or choosing a slow machine to increase workload.

Two approaches are possible among the studies that try to achieve multiple goals. The lexicographic approach searches for the schedule that meets the goals in a lexicographic order. The weighted-sum approach seeks for the schedule that achieves the highest scores of a linear combination of the goals.

A measure of the schedule varies from plant to plant, from single goal to multiple goals and from lexicographic approach to weighted-sum approach. This research aims to solve an optimization problem that achieves various goals subject to a set of feasible schedules that are generated for a set of demands on the basis of product data with resource flexibility and some WIP. The problem is called dynamic flexible scheduling (DFS) problem. It is flexible because it assigns resources in a work center to an operation, and because it responds to various goals. It is dynamic because it is requested to respond to any change in real time. The DFS problem is practical because it adopts the product data that is actually used in commercially available manufacturing planning software (such as SAP R/3 and SyteAPS).

NP-hard problems are problems for which there is no known polynomial algorithm, so that the time to find a solution grows exponentially in problem size (Hopp and Spearman, 2000). Job shop scheduling problem is a simplified version of DFS problem, which will be shown in Section 4, and it has been shown to be NP-hard by Croce et al. (1995) and

Al-Hakim (2001), hence DFS is also an NP-hard problem.

GA exhibits parallelism, contains certain redundancy and historical information of the past solutions. It is suitable for implementation on massively parallel architecture (Wang and Zheng, 2001), and it has been applied to a large number of complex search problems (Nearchou, 2004). GA does not rely on analytical properties of the function to be optimized, which makes them well suited to a wide class of optimization problems (Al-Hakim, 2001). However, in view of the randomness property of GA, there is no guarantee of reaching optimum solutions for most scheduling problems.

In this paper, genetic algorithm with MGG and demand crossover (MDGA) is proposed to solve DFS problem. MGG is an abbreviation of minimal generation gap that is a generation alternation model proposed by Yamamura et al. (1996), which keeps variety of chromosomes in a population while preventing the search process from local optima. Unlike one-point or two-point crossover, demand crossover is a new way proposed in this paper to exchange the genes that are related to some demands without violating the precedence constraints. Aytug et al. (2003) provided a review of the use of genetic algorithms to solve the production and operations management (POM) problems. The scheduling problem is one of them, but the optimization of DFS problem is not in those reviewed researches. In this sense, DFS problem is new; in addition practical and suitable for responding to changes agilely as compared to the job shop scheduling problem.

Section 2 introduces APPCS and how it responds to a change by an example. Section 3 provides a definition of DFS problem, and a formulation of the problem. Section 4 presents MDGA, and gives exhaustive search and a comparison with other GAs to demonstrate its correctness and effectiveness. Section 5 provides an insight into the performance of MDGA through an experiment and gives some advice on applying MDGA to solve DFS problem.

2. Agile production planning and control system (APPCS)

A product data is the data related to product design and manufacturing. The product data for APPCS contains part, bill-of-materials (BOM), routing, work center, and resource. Those terms are illustrated with a product part shown in Fig. 1a. A gray square in the product data shows a part. Finished product, assembly, and raw material are the types of part. This figure shows that it needs two pieces of assembly 'A' and two pieces of assembly 'B' to make one finished product 'F'. BOM is a term used to define such a request-supply relation.

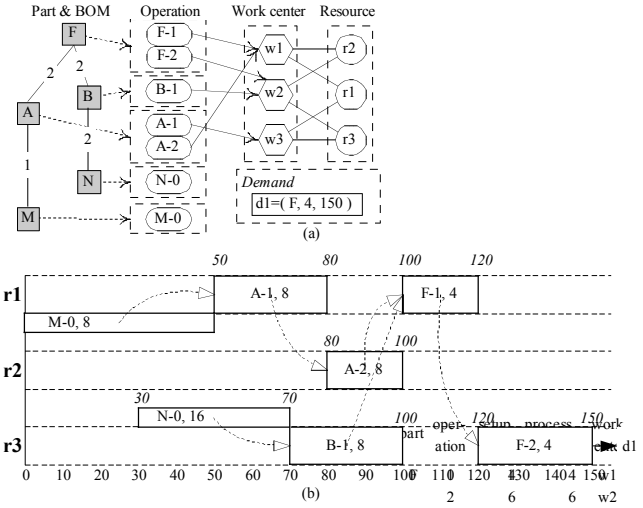


Fig. 1. (a) Product data of part 'F' and (b) a schedule for a demand 'd1'

A routing is a sequence of operations to make a part. A work center that enrolls some resources is assigned to an operation, and the operation will be processed by one of the resources. It takes some time to set up a resource before starting an operation. Setup time and processing time for processing a piece of part are estimated for an operation. However, the operation for procurement is processed without specifying a work center. As shown in Fig. 1a, finished part 'F' has two operations – 'F-1' and 'F-2'. Operation 'F-2' is processed at work center 'w2' in which resources 'r2' and 'r3' are stationed. It takes 6 time units to set up either resource, and 6 time units to process a part.

A demand is either a customer order or the result of forecasting. In Fig. 1b, a schedule generated for a demand 'd1' is illustrated with a Gantt chart. The demand requests 2 pieces of

finished product 'F' before due time 150. A rectangular bar in the chart shows a task. In a sense, a schedule is a set of tasks that are deployed in a Gantt chart. The dot line with a white arrow in the chart shows the precedence constraints that regulate the manufacturing sequence of tasks. According to the schedule, a task is released to the shop floor or it is passed on to the purchasing department for subsequent processing. According to Gantt chart shown in Fig. 1b, the first task (M-0, 8) should be lunched to a vender at time 0.

Assume a new demand 'd2', which requests 2 pieces of 'F' by time 180, arrives at time 30. According to APPCS, once an event causes any changes in a schedule to happen, all the planned tasks except for the in-processing ones are canceled and a new feasible schedule is plotted out again based on the in-processing tasks according to the updated conditions. When the in-processing task is finished, its output becomes a work-in-process (WIP).

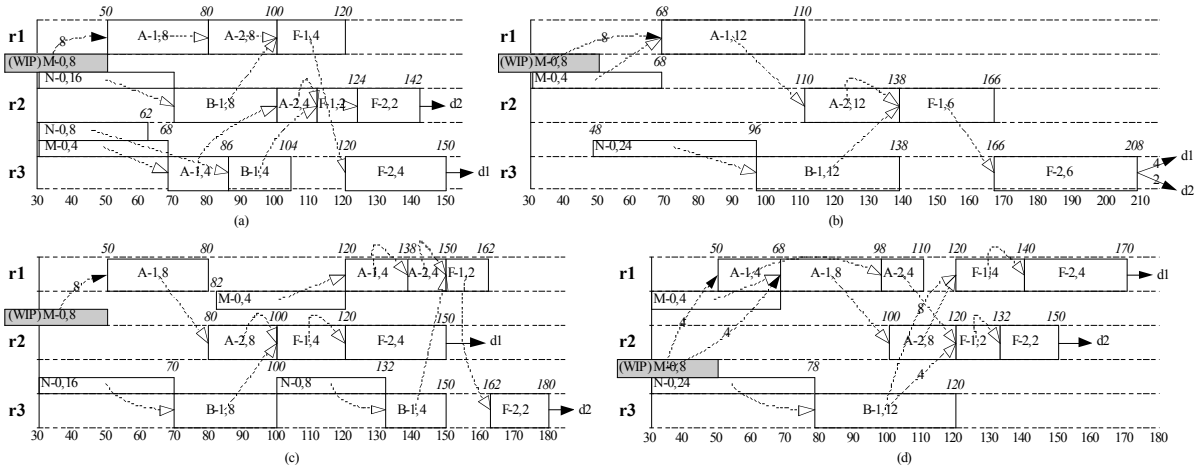


Fig. 2. A schedule achieves (a) minimum makespan, (b) minimum processing time, (c) maximum service level, and (d) a weighted-sum

Fig. 2a gives a new schedule for demands 'd1' and 'd2' based on the WIP (M-0, 8) whose planned finish time is 50. A dot line with a solid arrow shows that 8 pieces of the WIP are input to a downstream task (A-1, 8), which starts from time 50. Consequently, it is not necessary to generate a task for operation 'M-0'. This schedule achieves minimal makespan. Some plants may not satisfy with this schedule, because it indirectly causes a long processing

time (318) and tardiness (38).

Fig. 2b shows another schedule that achieves minimum total processing time (268), and the schedule in Fig. 2c attains maximum service level (100%), minimum earliness (0), and minimum tardiness (0). It is reasonable to process a group of identical operations together, to cut down on setup time of processing or on ordering costs of purchasing. The minimum total processing time of the schedule in Fig. 2b is achieved at the cost of service level (0%), tardiness (86) and makespan (178). Fig. 2d shows a schedule that compromise a goal with makespan of 140 and total processing time of 288 by applying the weighted-sum approach, where the minimum makespan and total processing time are 128 and 268, respectively.

In this manner, any change to a schedule will trigger APPCS to generate an improved, goal-oriented schedule recursively.

3. Dynamic flexible scheduling problem

3.1. Definitions and notations

Product data

The following notations are used in defining product data. An instance of the notation is provided following the description. The instance is drawn from the product data shown in Fig. 1a.

p_i	A part
$P=\{p_i\}$	The set of parts concerned; $P=\{$ F, A, B, M, N $\}$
$bm_{ij}=(p_i, p_j)$	An ordered pair indicates that part p_j is an immediate component of part p_i
$Bm=\{bm_{ij}\}$	The set of the ordered pairs bm_{ij} among parts in P; $Bm=\{$ (F, A), (F, B), (A, M), (B, N) $\}$
$Qty(bm_{ij})$	Quantity of p_j per unit of p_i; Qty(F, A)=2
$op_{ij}=(p_i, j)$	An ordered pair, called an operation, represents the jth processing step to make part p_i

$Nop(p_i)$ The number of operations of part p_i; $Nop(\mathrm{F})=2$

$Spt(op_{ij})$ Setup time of operation op_{ij}; $Spt(\mathrm{F}, 1)=4$

$Pct(op_{ij})$ Unit processing time of operation op_{ij}; $Pct(\mathrm{A}, 2)=2$

$Pcw(op_{ij})$ Processing work center of operation op_{ij}; $Pcw(\mathrm{B}, 1)=w2$

w_i A work center

$W=\{w_i\}$ The set of work centers; $W=\{w1, w2, w3\}$

r_i A resource

$R=\{r_i\}$ The set of resources; $R=\{r1, r2, r3\}$

$En(w_i)$ The set of resources enrolled in a work center w_i; $En(w_2)=\{r2, r3\}$

Recursively applying the request-supply relations defined in Bm, a hierarchy of parts is constructed. If each part in the hierarchy is replaced with its operations, then we get a hierarchy of operations.

$Rat(op_{im}, op_{jn})$ Number of parts necessary to be processed by an operation op_{jn} for a part by the successor operation op_{im} defined in the hierarchy of operations; it is defined as

$$Rat(op_{im}, op_{jn}) = \begin{cases} 1 & \text{if } p_i = p_j, m = n+1 \\ Qty(p_i, p_j) & \text{if } (p_i, p_j) \in Bm,\ m=1,\ n = Nop(p_i) \end{cases} \quad (1)$$

Demands

The following notations are used to denote the demands. The instance provided after the explanation is from Fig. 3, which shows a schedule in terms of the terminologies of the problem.

d_i A demand

$D=\{d_i\}$ A set of demands; $D=\{d_1, d_2\}$

$Rqq(d_i)$ Request quantity of demand d_i; $Rqq(d_1)=4$

$Rqp(d_i)$ Request part of demand d_i; $Rqp(d_2)=\mathrm{F}$

$Dut(d_i)$ Due time of demand d_i; $Dut(d_1)=150$

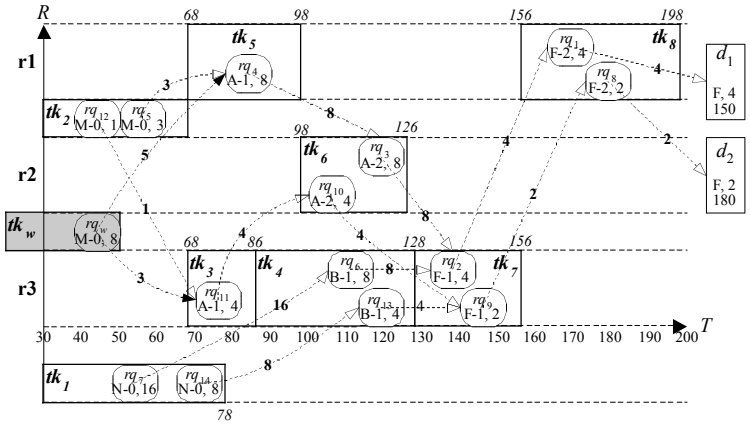


Fig. 3. A schedule in terms of terminologies of DFS problem

Operation requirements

An operation requirement is a request for an operation. The request is exploded either independently or dependently from a demand. 'Requirement' is used instead of 'operation requirement' for simplification. The following notations are used in defining requirements and their relations.

rq_i	A requirement
$Rq=\{rq_i\}$	A set of requirements exploded from demands D; $Rq=\{rq_i\}_{i=1..14}$
$Opr(rq_i)$	Operation of requirement rq_i; $Opr(rq_6)=$(B, 1)
$Net(rq_i)$	Quantity (net requirement) of requirement rq_i; $Net(rq_{10})=4$
$e_{ij}=(rq_i, rq_j)$	An ordered pair showing precedence relation that rq_j must be processed before rq_i
$E_{rq}=\{e_{ij}\}$	A set of directed edges representing precedence among requirements Rq; $E_{rq}=\{(rq_1, rq_2), (rq_2, rq_3), (rq_3, rq_4), (rq_4, rq_5), (rq_2, rq_6), (rq_6, rq_7),\ldots, (rq_{13}, rq_{14})\}$
$G_{rq}=(Rq, E_{rq})$	A directed acyclic graph of the requirements Rq
$Pd(rq_i)$	The set of immediate predecessors of rq_i; $Pd(rq_9)=\{rq_{10}, rq_{13}\}$
$Sc(rq_i)$	The immediate successor of rq_i; $Pd(rq_4)=rq_3$

$Met(d_i)$ The requirement planned to meet demand d_i; $Met(d_2)=rq_8$

The requirements in Rq must be generated according to the hierarchy of operations. Let $(rq_i, rq_j)\in E_{rq}$ be arbitrary, and assume that operation $Opr(rq_i)=(p_i, m)$ and $Opr(rq_j)=(p_j, n)$. If $p_i = p_j$, then $m=n+1$; otherwise $(p_i, p_j)\in Bm$, $m=1$, and $n=Nop(p_j)$. Requirements are generated to meet the request of all demands. For a demand d_i, there must be one and only one requirement, $rq_i\in Rq$, such that $Sc(rq_i)=\varnothing$, $Opr(rq_i) = (Rqp(d_i), Nop(Rqp(d_i)))$ and $Rqq(d_i)=Net(rq_i)$ hold.

Tasks

A task is an operation processed within a period of time by a resource, which is generated to meet one or more requirements. The following notations are used for defining tasks and the relation with requirements. Notice that for a set A, $|A|$ represents the number of elements in A.

tk_i A task

$Tk=\{tk_i\}$ A set of tasks scheduled to meet requirements Rq, $|Tk|\le|Rq|$; $Tk=\{tk_i\}_{i=1..8}$

$Rsc(tk_i)$ The resource assigned to process task tk_i; $Rsc(tk_3)=r3$

$Sta(tk_i)$ The start time of task tk_i; $Sta(tk_3)=68$

$Fin(tk_i)$ The finish time of task tk_i; $Fin(tk_3)=86$

$Tq(tk_i)$ A set of requirements scheduled to form a task tk_i; $Tq(tk_4)=\{rq_6, rq_{13}\}$

$Qt(rq_i)$ The task of requirement rq_i; $Qt(rq_6)=tk_4$

Sst Scheduling start time; $Sst=30$

$Est(rq_i)$ The earliest time at which rq_i can be started; $Est(rq_3)=98$, $Est(rq_{10})=86$

$Lft(rq_i)$ The latest time which rq_i must be completed; $Lft(rq_3)=128$, $Lft(rq_{10})=128$

The earliest and latest times for a requirement are defined as

$$Est(rq_i)=\begin{cases} Sst & \text{if } Pd(rq_i)=\varnothing \\ \max\limits_{rq_j\in Pd(rq_i)} Fin(Qt(rq_j)) & \text{otherwise} \end{cases} \quad (2)$$

and

$$Lft(rq_i) = \begin{cases} \infty & \text{if } Sc(rq_i) = \varnothing \\ Sta(Qt(Sc(rq_i))) & \text{otherwise} \end{cases}. \quad (3)$$

WIP

A rigorous definition of 'work-in-process' that a requirement whose successor is canceled by some changes is used in this paper, instead of the general one that a requirement whose task is in-processing. This is because that scheduling start time *Sst* might be far later than notification time of a change, and some in-processing tasks might have been completed.

Wp The set of work-in-processes; $Wp = \{ rq_w \}$

$Qw(rq_i)$ A set of WIP that was allocated to a requirement rq_i; $Qw(rq_4) = \{ rq_w \}$

$Wq(rq_w)$ A set of requirements to which a WIP rq_w is allocated; $Wq(rq_w) = \{ rq_4, rq_{11} \}$

$Alq(rq_i, rq_w)$ The quantity of WIP rq_w allocated to requirement rq_i; $Alq(rq_4, rq_w) = 5$

Constraints on scheduling

[C1] Operation consistency for a task: The requirements with identical operation can be combined to form a task. That is, the following constraint should hold.

$$(\forall tk_i \in Tk)\ (\forall rq_i, rq_j \in Tq(tk_i))\ \ Opr(rq_i) = Opr(rq_j) \quad (4)$$

[C2] Total processing time: The total processing time of a task, i.e. the difference between finish time and start time, equals to the sum of setup time and the product of unit processing time and the sum of quantities of the contained requirements. It is

$$(\forall tk_i \in Tk)\ \ (\exists rq_i \in Tq(tk_i))\ \ Fin(tk_i) - Sta(tk_i) = Spt(Opr(rq_i)) + Pct(Opr(rq_i)) \times \sum_{rq_j \in Tq(tk_i)} Net(rq_j). \quad (5)$$

[C3] Resource flexibility: If there is a resource assigned to a task, then it must be enrolled in the work center that is assigned to the operation of requirements satisfied by the task. That is,

$$(\forall tk_i \in Tk)\ (\exists rq_i \in Tq(tk_i))\ \ Rsc(tk_i) \in En(Pcw(Opr(rq_i))). \quad (6)$$

[C4] Precedence of tasks: The constraint that a task cannot be scheduled to start earlier than the latest earliest start time and to finish later than the earliest latest finish time among its requirements is denoted by

$$(\forall tk_i \in Tk)\ \ Sta(tk_i) \geq \max_{rq_i \in Tq(tk_i)} Est(rq_i) \text{ and } Fin(tk_i) \leq \min_{rq_i \in Tq(tk_i)} Lft(rq_i). \quad (7)$$

[C5] Finite loading on resource: The finite loading constraint of a resource is denoted by

$$(\forall tk_i,\ tk_j \in Tk)\ \text{if}\ \ Rsc(tk_i) = Rsc(tk_j),\ \text{then}\ [Sta(tk_i), Fin(tk_i)) \cap [Sta(tk_j), Fin(tk_j)) = \varnothing,$$

where $[t_1, t_2)$ means an interval of time from t_1 to t_2. (8)

[C6] WIP allocation: A WIP (planned requirement) can be used by a requirement if it has the same operation with the WIP, and if the WIP is completed before the successor of the requirement starts. That is,

$$(\forall rq_i \in Rq)(\forall rq_w \in Qw(rq_i))\ \ Opr(rq_i) = Opr(rq_w) \text{ and } Fin(Qt(rq_w)) \leq Sta(Qt(Sc(rq_i))). \quad (9)$$

[C7] Total quantity of WIP: The total allocated quantity of a WIP among requirements cannot exceed quantity of the WIP. That is,

$$(\forall rq_w \in Wp)\ Net(rq_w) \geq \sum_{rq_i \in Wq(rq_w)} Alq(rq_i, rq_w). \quad (10)$$

[C8] Net requirement: For any $(rq_i,\ rq_j) \in E_{rq}$, quantity of rq_j is calculated by subtracting WIP allocations from the gross requirement requested by rq_i. That is,

$$(\forall (rq_i,\ rq_j) \in E_{rq})\ Net(rq_j) = Net(rq_i) \times Rat(Opr(rq_i), Opr(rq_j)) - \sum_{rq_w \in Qw(rq_i)} Alq(rq_j, rq_w). \quad (11)$$

Dynamic flexible scheduling problem

Denote evaluation functions of makespan by EV_{mks}, service level by EV_{svc}, and tardiness by EV_{tds} for a schedule. Dynamic flexible scheduling (DFS) problem is defined as:

$$\text{Minimize}\ \ EV_{mks} = \max_{tk_i \in Tk}(Fin(tk_i)) - \min_{tk_i \in Tk}(Sta(tk_i)), \quad (12)$$

$$\text{Maximize}\ \ EV_{svc} = \left(\sum_{d_i \in D} I(Fin(Qt(Met(d_i))) \leq Dut(d_i))\right)/|D|, \text{ where } I: \{T, F\} \rightarrow \{1, 0\}, \text{ or} \quad (13)$$

$$\text{Minimize}\ \ EV_{tds} = \sum_{d_i \in D} \max\{Fin(Qt(Met(d_i))) - Dut(d_i), 0\}, \quad (14)$$

Subject to the following eight constraints.

[C1] Operation consistency for a task

[C2] Total processing time

[C3] Resource flexibility

[C4] Precedence of tasks

[C5] Finite loading on resource

[C6] WIP allocation

[C7] Total quantity of WIP

[C8] Net requirement

3.2. Problem formulation

Requirement arrangement, requirement aggregation, resource assignment, WIP allocation, and scheduling alternatives are the steps to formulate DFS problem in a systematic way.

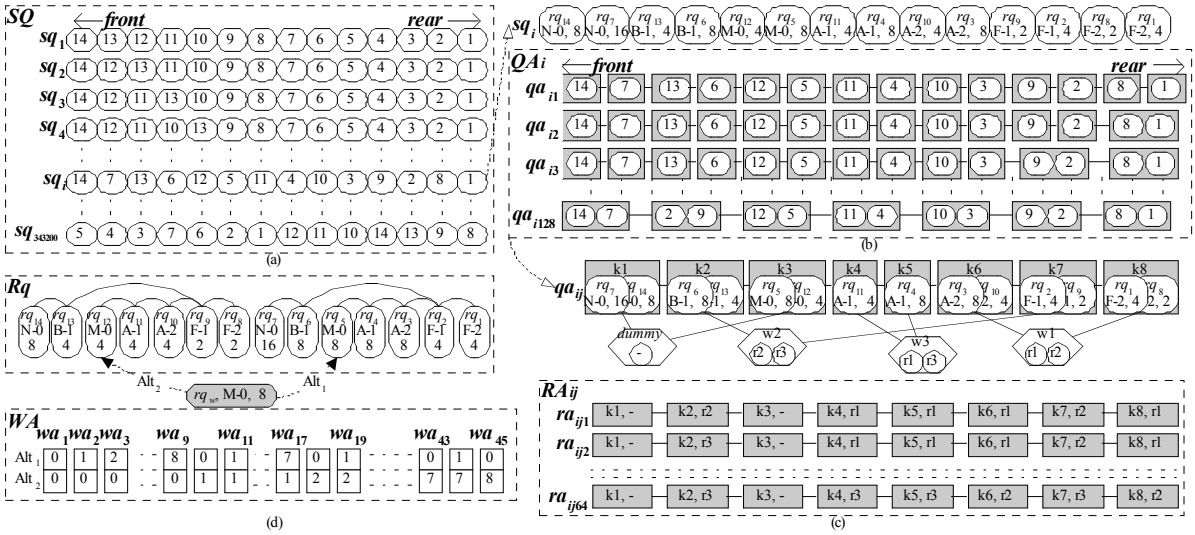


Fig. 4. (a) Legal sequences of requirements, (b) sequences of baskets, (c) possible resource assignments for a sequence of baskets, (d) possible WIP allocations for a set of requirements

(1) Requirement arrangement: Production planning and scheduling assigns available capacity (a time interval) of a resource to requirements in Rq. To solve the conflict caused when more than one requirement requests for the same period of time from a resource, these requirements are arranged to a sequence $\langle rq_i \rangle_{i=1..|Rq|}$ ($rq_i \in Rq$) and the capacity of resource is assigned to the requirements in order of the sequence. Let $SQ = \{ sq_1, sq_2, \ldots, sq_k \}$ be a set of all legal sequences on the requirements in Rq, where a sequence of the requirements is said to be legal if the order of the requirements doesn't violate the precedence

constraints E_{rq}. Fig. 4a shows the set SQ.

(2) Requirement aggregation: The requirements with the same operation that are located adjacent to each other in a sequence can be grouped together into a basket. A basket is a basic unit of scheduling and the requirements in a basket will be scheduled together to form a task. A basket is for a requirement if either side of the requirement does not have any requirement with identical operation. The aggregation of adjacent requirements without shifting their position in a sequence complies with the precedence constraints E_{rq}. Denote a set of sequences of baskets by $QA_i = \{ qa_{i1}, qa_{i2}, \ldots, qa_{im} \}$ on a sequence of requirements $sq_i \in SQ$. Fig. 4b shows the possible cases of QA_i for a $sq_i \in SQ$ in Fig. 4a.

(3) Resource assignment: A work center is assigned to an operation except operations that need to be planned lead time for procurement. To keep it simple, we assume that such an operation is assigned to a dummy work center. A basket, including at least one requirement, inherits work center from the requirements, and one of the resources in the work center is assigned to the basket for scheduling. For a sequence of requirements $sq_i \in SQ$, and for a sequence of baskets $qa_{ij} \in QA_i$, let $RA_{ij} = \{ ra_{ij1}, ra_{ij2}, \ldots, ra_{ijk} \}$ be a set of sequences of resource-assigned baskets. Fig. 4c shows some instances of resource assignment for a sequence of baskets in Fig. 4b. Dummy resource is assigned to baskets k1 and k3, because they are the aggregation of procurement requirements.

(4) WIP allocation: Quantity of WIP can be allocated to the requirements of the same operation, as shown in [C7], in the new scheduling run. Fig. 4d shows the alternative ways to allocate 8 units of WIP rq_w to rq_5 and rq_{12}. Let WA be a set of the possible WIP assignments from WIP in Wp to requirements in Rq.

(5) Scheduling alternatives: $SA = \{fs, bs\}$ is a function set of two scheduling alternatives - forward scheduling and backward scheduling. Backward scheduling generates a schedule backwardly from due time of a demand, while forward scheduling does it forwardly from

the scheduling start time *Sst*.

```
/* Terms denote baskets together with their relations with requirements
Bk={ bk_i }        A set of the resource-assigned baskets
⟨bk_i⟩_{i=1..|Bk|}  A sequence of baskets on Bk
Bq( bk_i )         A set of requirements in a basket

/* Production planning and scheduling
Production_ Planning_and_Scheduling ( Sequence of baskets: ⟨bk_i⟩ )
(01) IF scheduling alternatives = 'forward scheduling'
(02) THEN DO Forward_Scheduling (Sequence of baskets: ⟨bk_i⟩);
(03) ELSE DO Backward_Scheduling (Sequence of baskets: ⟨bk_i⟩);

/* Backward scheduling of a sequence of baskets
Backward_Scheduling ( Sequence of baskets: ⟨bk_i⟩ )
(04) FOR each basket bk_i∈Bk in a reverse order of ⟨bk_i⟩
(05)     Calculate quantity of each requirement in bk_i by [C8]
         in the confines of [C6];
(06)     Generate a task tk_n for all requirements in bk_i due to [C1];
(07)     Get resource rs assigned to basket bk_i due to [C3];
(08)     Calculate total processing time tpt of tk_n by [C2];
(09)     Get a set of intervals Itv of resource rs whose length ≥ tpt
         and finish time ≤ min{ Lft( rq ) | rq∈Bq(bk_i) } due to [C4];
(10)     Pick an interval of the latest finish time fin from Itv, and
         reserve capacity [fin − tpt, fin) of rs for tk_n due to [C2].
(11) ENDFOR
(12) IF any task in the Gantt chart starts before Sst
     THEN DO Forward_Scheduling ( Sequence of baskets: ⟨bk_i⟩);

/* forward scheduling of a sequence of baskets
Forward_Scheduling (Sequence of baskets: ⟨bk_i⟩ )
(13) FOR each basket bk_i∈Bk in a reverse order of ⟨bk_i⟩
(14)     Calculate quantity of each requirement in bk_i by [C8];
(15) ENDFOR
(16) FOR each basket bk_i∈Bk in the order of ⟨bk_i⟩
(17)     Generate a task tk_n for all requirements in bk_i due to [C1];
(18)     Get resource rs assigned to basket bk_i due to [C3];
(19)     Calculate total processing time tpt of tk_n by [C2];
(20)     Get a set of intervals Itv of resource rs whose length ≥ tpt,
         start time ≥ max{ Est( rq ) | rq∈Bq(bk_i) } due to [C4], and
         start time ≥ max{ Fin(Qt(rq_w)) | rq_w∈Qw(Bq(bk_i)) } due to
[C6];
(21)     Pick an interval of the earliest start time sta from Itv, and
         reserve capacity [sta, sta+tpt) of rs for tk_n due to [C2];
(22) ENDFOR
```

Fig. 5. Procedures of production planning and scheduling

The ways to calculate the total number of cases in requirement arrangement, requirement aggregation, and WIP allocation are shown in Appendix A, B, and C, respectively. Domain of

DFS problem formulated in a systematic way can be denoted by $SA \times WA \times \bigcup_{i=1}^{|SQ|} \bigcup_{j=1}^{|QA_i|} RA_{ij}$. Production planning and scheduling is to transform a sequence of resource-assigned baskets with respective WIP allocation and the specification of a scheduling alternative into a set of tasks, which can be deployed on a Gantt chart. One basket, including a set of requirements, is converted to a task.

The procedure of production planning and scheduling is shown in Fig. 5. Lines from (01) to (03) show that a chromosome running forward scheduling or backward scheduling is determined by the scheduling alternative. Lines from (04) to (11) show backward scheduling runs net requirement planning together with scheduling in a sequence one by one from the rear basket back to the front one. In line (05), quantity (net requirement) of each requirement in a basket is planned by deducting effective quantity of WIP allocation (due to [C6]) from gross requirement of the successor according to [C8]. Lines (06) and (07) show that a task is generated for a basket and the resource for the task is brought from the basket. Referring to [C2], total processing time of a task is calculated in line (08). In line (09), available intervals of the resource enough and in time for the processing time are gathered. Finish time of the intervals cannot be later than the start time of the successor requirements. The interval with the latest finish time among the intervals is selected and occupied with the processing time of the task as denoted in line (10). Finally, as shown by line (12), if the schedule by backward scheduling starts before scheduling start time *Sst*, then forward scheduling is triggered to generate a feasible schedule from *Sst*.

Forward scheduling plans net requirement from line (13) to (15), then runs scheduling from the front basket to the rear one as listed from line (16) to (21). The net requirement planning is similar to backward scheduling with the exception that all allocated WIP is forced to be used in offsetting the gross requirement as shown in line (17). However, finish times of the WIP must be taken into consideration in determining the earliest start time of the task. As

shown in line (20), the earliest start time forces a new task to start after not only the finish times of the predecessor requirements but also the allocated WIP.

The result of production planning and scheduling of a sequence of baskets appeared in Fig. 4 is shown in Fig. 6, in which '*qty*', '*tpt*', '*lft*', and '*est*' are total net requirements, total processing time, latest finish time, and earliest start time of a basket, respectively, for deploying a task in the Gantt chart. As shown by Fig. 6a, backward scheduling plans from basket k8 to k1. A WIP rq_w that ends in time 50 is allocated to requirements rq_5 and rq_{12}, but the WIP is not in time for tasks tk_3 and tk_5, hence the allocation is unusable. Consequently, the schedule by backward scheduling is not feasible because it starts before the scheduling start time *Sst*. Forward scheduling is done, accordingly. Fig. 6b shows the resultant feasible schedule run by forward scheduling starting from $Sst = 30$, while the allocated WIP is used, hence less material needs to be purchased.

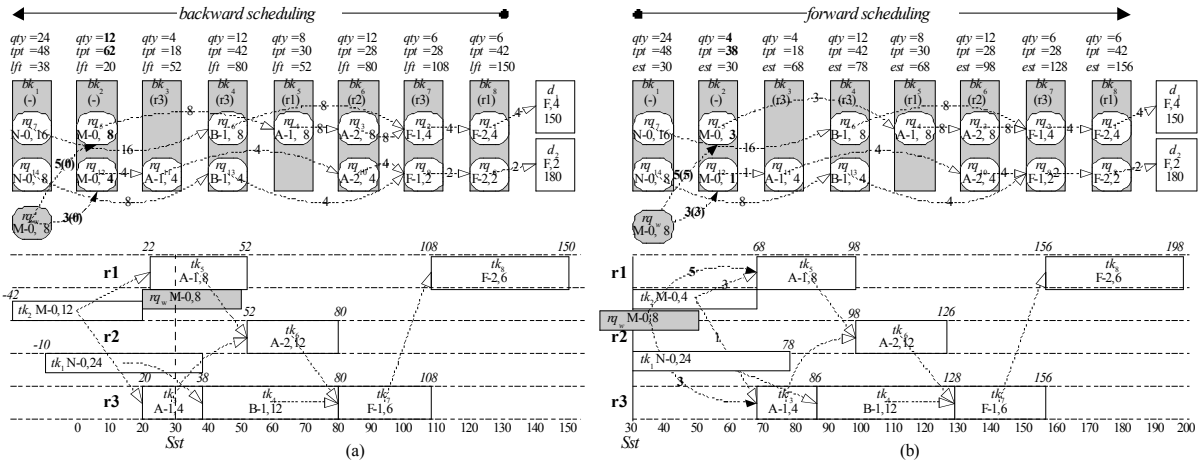


Fig. 6. Results of (a) backward scheduling, and (b) forward scheduling

4. Genetic algorithm with MGG and demand crossover (MDGA)

This paper proposes a specific genetic algorithm, called MDGA, to solve DFS problems. A chromosome acts as information carrier through the processes of MDGA. It joins the reproduction process to propagate its offspring by demand crossover and mutation. Then, the offspring's fitness values are measured to compete with those of other chromosomes by

minimal generation gap (MGG) generation alternation process to decide whether they can be promoted to the next generation. If lost, it is abandoned to have more room for a new chromosome. The processes of reproduction and selection are repeated until all termination conditions are satisfied. The correctness and effectiveness of MDGA will be examined by exhaustive search and a comparison with other GAs.

4.1. Encoding

To encode a chromosome is to represent an instance of domain of DFS problem. A chromosome is a combination of components, called genes. We encode a gene with a requirement, and a chromosome with a sequence of requirements.

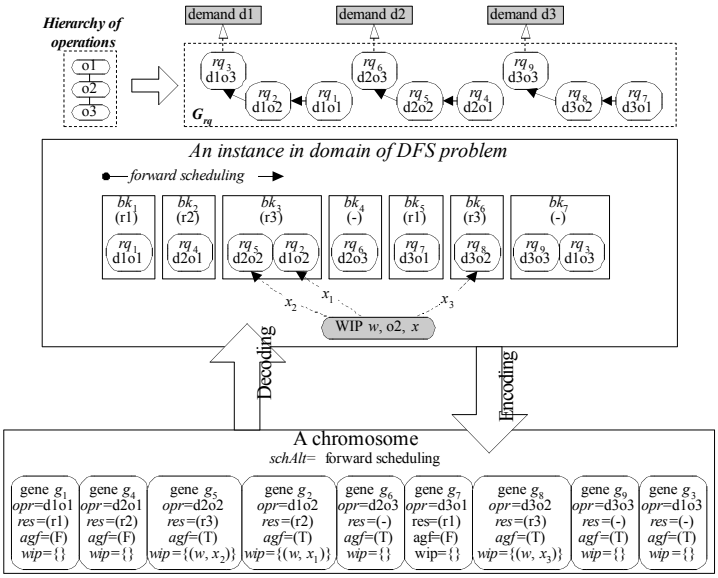


Fig. 7. The mapping of DFS problem to the GA encoding

A gene has four attributes: operation, aggregation flag, resource, and WIP allocation. Aggregation flag and operation advise whether the gene is capable of aggregating with other genes. A chromosome has an attribute for scheduling alternative, which suggests whether the chromosome should run forward scheduling or backward scheduling. Fig. 7 shows the encoding corresponding to the DFS problem, where '*opr*' indicates the operation, '*res*' the assigned resource, '*agf*' the aggregation flag, '*wip*' the WIP allocations of gene, and '*schAlt*' the scheduling alternative.

If several genes have the same operation, 'True' aggregation flags, and are adjacent each other in a chromosome, then they are grouped together in a basket. Subsequently, the responsible resource for a group of genes is randomly selected among resources of the genes. As a result, a chromosome can be decoded back to a sequence of baskets.

4.2. Initialization

At first a set of chromosomes is randomly generated to form a population. An initialization procedure is proposed in Fig. 8a to generate a legal chromosome for the initial population, where the lines from (02) to (09) are a loop for sampling a sequence of genes without violating the precedence constraints. The key to that is appending the predecessor requirements to the sampling pool Q in line (04) immediately after a requirement is removed from Q as shown in line (03). For a gene, a resource is randomly skimmed off among the resources selected by applying [C3] in line (06), and the aggregation flag is set in line (07). At the end, the quantity of WIP, if any, is distributed to suitable genes randomly according to rules [C6], [C7], and [C9] in line (10). From line (11) to (12), a chromosome is generated to contain the sequence of genes, and the scheduling alternative of the chromosome is set to be either forward scheduling or backward scheduling.

/* *Generate an initial population of chromosomes*	/* *Generation alternation with rMGG*
(a) Initialization ( )	(b) rMGG (m, n, k)
(01) Put the last requirement in Rq into a queue Q;	(01) Generate m chromosomes as initial population P;
(02) WHILE Q is not empty	(02) FOR each generation UNTIL nth generation
(03) Remove *any* requirement rq_i from Q;	(03) Remove 2 chromosomes $\{x, y\}$ from P;
(04) Add predecessor requirements $Pd(rq_i)$ to Q;	(04) Apply N-demand crossover on $\{x, y\}$, and get $\{x', y'\}$;
(05) Create a gene gn for rq_i;	(05) Evaluate x' and y';
(06) Choose a resource in $En(Pcw(Opr(rq_i)))$ for gn due to [C3];	(06) Choose the best fit from $\{x, y, x', y'\}$ as b;
(07) Set aggregation flag of gn to be 'True' or 'False';	(07) Choose *any* one from $\{x, y, x', y'\} - \{b\}$ as a;
(08) Append gn to a queue of genes G;	(08) Put a and b back to P;
(09) END WHILE	(09) IF random number < k
(10) Allocate quantity of WIP to genes in G randomly according to [C6], [C7], [C9];	(10) Select a chromosome ch from P;
(11) Designate a chromosome C contains G;	(11) Apply shift mutation to ch, and evaluate it;
(12) Assign a scheduling function in $\{'fs', 'bs'\}$ to C;	(12) END IF
	(13) END FOR

Fig. 8. Procedures of (a) initialization, and (b) generation alternation with rMGG

4.3. Reproduction

Two methods, crossover and mutation, are used in MDGA to reproduce new chromosomes. In general, crossover operator randomly selects two chromosomes from a population, exchanges some genes of them, and reproduces two new chromosomes. Mutation operator randomly selects a chromosome from the population, reverses some data, and then puts it back to the population. For DFS problem, both operators must comply with the precedence constraint when reproducing a new sequence of genes. The crossover and mutation of MDGA are explained as follows.

(1) N-demand crossover

The genes in a chromosome with common attributes form a sub-chromosome. The genes sharing the same resource are competitors, while the genes belonging to the same demand are partners. Besides, the genes reside in a sub-chromosome with their positions in the chromosome might provides some valuable information on solving DFS problem.

After selecting 2 chromosomes from the population, N-demand crossover begins with choosing N demands from *D* randomly, and then identifies the genes belonging to those demands in both chromosomes. Finally, it exchanges the genes from N demands in a chromosome with the genes in another chromosome.

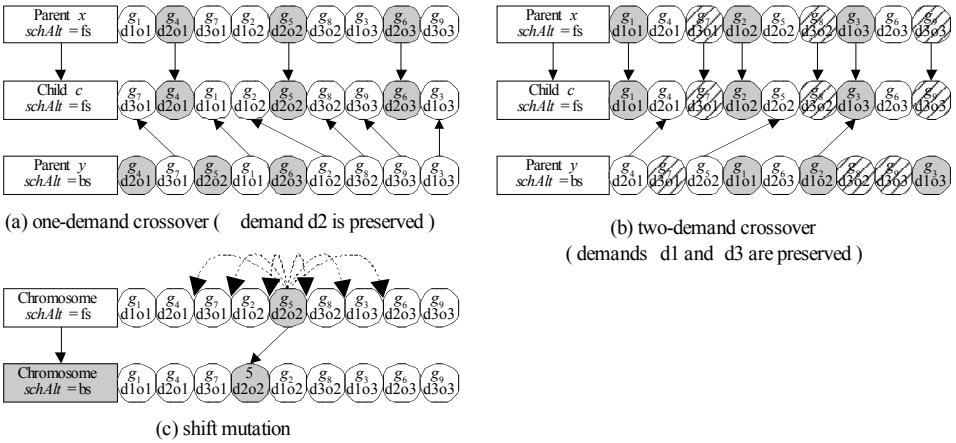


Fig. 9. The reproduction operators

Fig. 9a shows how one-demand crossover swaps genes $\{g_4, g_5, g_6\}$, which belong to demand d2 as shown in Fig. 7, between parent chromosomes $\{x, y\}$ to make a child chromosome c. Similarly, two-demand crossover swaps genes from demands $\{\mathrm{d1}, \mathrm{d3}\}$ in a chromosome with the genes from the same demands in another chromosome, as shown in Fig. 9b.

The crossover operator exchanges not only the sequence of genes in a chromosome but also the embedded information including aggregation flag, resource, and WIP allocations. If the sum of the WIP allocations violates the constraint [C7] after crossover, deduct the surplus or replenish the shortage from the WIP allocations.

In order to consider the general cases, assume that the selected N demands contain a set of genes $G=\{g_1, g_2, \ldots, g_n\}$ in a chromosome. The chromosome is a sequence of genes denoted by $x=G_1<\{g_1\}<G_2<\ldots<\{g_n\}<G_{n+1}$, where '<' means the precedence relation in a chromosome, and G_1, G_2,…, G_{n+1} are sets of sub-chromosomes; on the same assumption, the same genes in chromosome y can be denoted by $y=H_1<\{g_1\}<H_2<\ldots<\{g_n\}<H_{n+1}$.

Define the crossover of x and y as $x \otimes y = (x \downarrow (\bigcup_{i=1}^{n+1} G_i)) \uparrow (y \downarrow G)$, where '$Q\downarrow S$' means taking genes in set S away from sequence Q but remaining the position of other genes unchanged, and '$Q\uparrow P$' means filling up the empty position in sequence Q with genes in a sequence P. However, the crossover of x and y, $x\otimes y$, is not necessarily equal to $y\otimes x$. The N-demand crossover operator abides by precedence constraints, because the genes in sequence $\langle g_i \rangle_{i=1}^{n}$ and ($H_1<H_2<\ldots<H_{n+1}$) belong to different demands.

(2) Shift mutation

A single gene is chosen randomly from a chromosome, and then inserted into a random position after its preceding genes and before its succeeding gene as shown in Fig. 9c. Besides, resource and aggregation flag of the gene, and scheduling alternative of the chromosome are randomly given a new value. The WIP allocation to the gene, if any, is set to 0, and the

mismatch caused by shift mutation invokes a process to redistribute WIP quantity among those genes sharing the WIP.

4.4. Generation alternation

Generation alternation models are important to provide controls on searching process. Existing works have often used SGA (simple or standard GA) for a fixed generation alternation model. Yamamura et al. (1996) proposed a roulette minimal generation gap (rMGG) as an extension of MGG. The MDGA applies rMGG to generation alternation.

Fig. 8b lists the steps of generation alternation of MDGA. Line (01) shows the generation of initial population; from line (03) to (08), the N-demand crossover operator; and from line (10) to (11), the shift mutation operator. The crossover operator selects two chromosomes from the population as shown in line (03), reproduces two descendants in line (04), evaluates them in line (05), chooses the best one among the four chromosomes in line (06) and any one from the remains in line (07), and puts the two chromosomes back to the population with replacement in line (08). In this figure, m denotes the population size, n the number of generations, and k the mutation rate.

The main difference between SGA and MDGA is that SGA reproduces all the offspring at a generation then carries out the tournament selection from the whole population, while MDGA executes roulette selection from the 4 chromosomes immediately after reproduction. MDGA keeps variety of chromosomes in a population, prevents the search process from rushing into local optima.

4.5. Correctness and effectiveness

(1) Exhaustive search

Exhaustive search (ES) is to examine all the possible elements in domain of DFS problem to find the best solution. As shown in Table 1, according to 4 sets of demands, four problems based on the product data shown in Fig. 1a, and 8 units of WIP M-0 are prepared. These problems and their results by ES and MDGA are shown in Table 1.

No	Demand set (part, qty, due)	DFS problem Domain size	Time	Result	E_{mks}	E_{prt}	E_{tds}	E_{svc}
1	d1=(F, 4, 150)	5,760	3 sec	ES	120	170	0	100
				MDGA (sn[1])	120 (1)	170 (3)	0 (1)	100 (1)
2	d1=(F, 4, 150) d2=(A, 4, 120)	25,021,440	3.1 hrs.	ES	120	212	0	100
				MDGA (sn)	120 (44)	212 (42)	0 (579)	100 (35)
3	d1=(F, 4, 150) d2=(A, 4, 180) d3=(B, 4, 80)	4,149,596,160	21.4 days	ES	120	248	0	100
				MDGA (sn)	120 (34)	248 (475)	0 (338)	100 (562)
4	d1=(F, 4, 150) d2=(F, 2, 180)	115,142,123,520	49.5 years	ES[2]	126	268	0	100
				MDGA (sn)	120 (633)	268 (1,162)	0 (25)	100 (39)

1 'sn' means number of times of the scheduling run when the best value is found by MDGA.
2 The result of running exhaustive search for about 2 months.

Table 1. Result of the exhaustive search

We accomplished the exhaustive search of problem 1, 2, and 3. Problem 4 had been tried for two months on a PC, while the best value found by ES during the search is even worse than that of MDGA. The correctness of MDGA is proved by that it achieves the optimum value identical to the result of ES. The computer executing ES and MDGA can process about 2000 chromosomes per second. The fact that MDGA reaches the optimum value in less than a second gives an account of the efficiency.

(2) Benchmark

Job shop scheduling (JSS) problem is a subset of DFS problem. Moreover, JSS problem is a restricted DFS problem. If we do not use bill of materials, routing flexibility, WIP, or setup time, if we specify only forward scheduling as the scheduling alternative, and if we select makespan as the evaluation function, then we have a JSS. In other words, it is hardly to produce a practical schedule by solving JSS where the use of product data is inevitable.

A benchmark of some famous JSS problems is used to compare with the work of Croce (1995), who proposed an encoding based on preference rules and an updating step which speeds up the evolutionary process. The problems whose identification starts with 'MT' are from Muth & Thompson, and with 'LA' are from Lawrence, according to Croce (1995).

It is not appropriate to compare the performance of GA on the basis of time, since the experiments are carried out on different computers with different operating system and implemented by using different programming languages with different skill. As MGG has

different definition of a generation with SGA, generation is not adequate for comparison either. Croce measured the performance on the basis of the number of chromosomes generated during a run. In a similar way, we count the times of scheduling as the basis of comparison.

The number of N-demand crossover (NDC) is set to the maximum degree (half of the number of demands). No mutation is set in the benchmark. Table 2 shows the performance of MDGA where the number of scheduling (SCH) is used as the termination parameter. SCHs are set to 10000, 30000, and 60000, respectively. Croce's result is shown for comparison, which uses 30000 chromosomes. Two population sizes (POP) are set: POP=50 in SCH=10000 for faster termination, and POP=100 in SCH=30000 and SCH=60000 for slower termination. If MDGA achieves the best value so far (OPT) by applying POP=50 to some easy problem, then the test for POP=100 is omitted. The best makespan shown in the table is selected over five runs, and so is the average makespan.

Problem	*n*	*m*	OPT[1]	NDC[2]	MDGA						Croce	
					POP=50		POP=100				POP=300	
					SCH=10000		SCH=30000		SCH=60000		SCH=30000	
					Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.
MT06	6	6	55	3	55	55.0	–	–	–	–	55	55.0
MT10	10	10	930	5	955	965.2	939	949.0	939	948.4	946	965.2
MT20	20	5	1165	10	1176	1193.4	1174	1178.0	1165	1172.2	1178	1199.0
LA01	10	5	666	5	666	666.0	–	–	–	–	666	666.0
LA06	15	5	926	7	926	926.0	–	–	–	–	926	926.0
LA11	20	5	1222	10	1222	1222.0	–	–	–	–	1222	1222.0
LA16	10	10	945	5	967	979.0	959	973.6	946	963.0	979	989.0
LA21	15	10	1048	7	1074	1098.8	1066	1077.4	1055	1071.2	1097	1113.6
LA26	20	10	1218	10	1281	1294.8	1220	1230.8	1218	1226.6	1231	1248.0
LA31	30	10	1784	15	1784	1784	–	–	–	–	1784	1784
LA36	15	15	1268	7	1336	1339.4	1305	1312.0	1297	1306	1305	1330.4

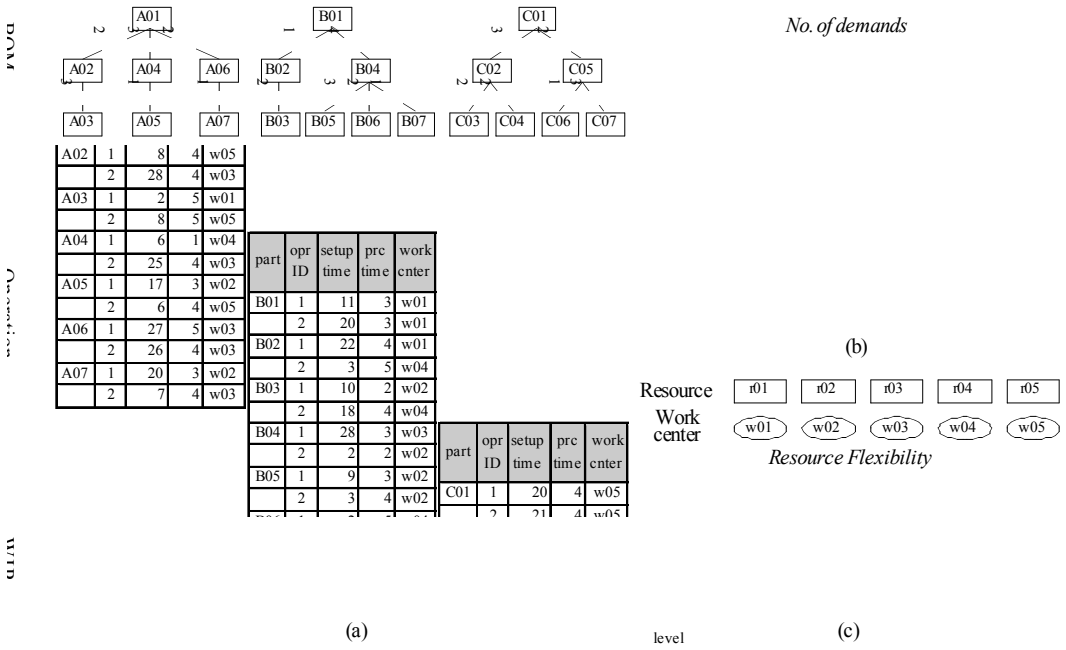
1. 'OPT' means the best value found so far by the heuristic researches.
2. 'NDC' means number of demand crossover using in the benchmark.

Table 2. Comparison of MDGA with Croce's GA

By observing Table 2, MDGA is not a bad method for solving JSS problem. Furthermore, its ability overcomes simple JSS solvers, in the sense that MDGA provides a way to handle practical product data and then is able to produce feasible schedule.

5. Experimental analysis

An experiment is conducted to investigate the factors and mutual effects on applying genetic algorithms with MGG and demand crossover (MDGA) to dynamic flexible scheduling (DFS) problem. Population size 50 is set, and the number of N-demand crossover (NDC) is set to be $0.5|D|$. The product data used in the experiment is shown in Fig. 10a. The factors in DFS problem that might have influence on performance of MDGA are the number of demand, routing flexibility, and evaluation functions.



A02	1	8	4	w05
	2	28	4	w03
A03	1	2	5	w01
	2	8	5	w05
A04	1	6	1	w04
	2	25	4	w03
A05	1	17	3	w02
	2	6	4	w05
A06	1	27	5	w03
	2	26	4	w03
A07	1	20	3	w02
	2	7	4	w03

part	opr ID	setup time	prc time	work cnter
B01	1	11	3	w01
	2	20	3	w01
B02	1	22	4	w01
	2	3	5	w04
B03	1	10	2	w02
	2	18	4	w04
B04	1	28	3	w03
	2	2	2	w02
B05	1	9	3	w02
	2	3	4	w02

part	opr ID	setup time	prc time	work cnter
C01	1	20	4	w05
	2	21	4	w05

Fig. 10. Experiment data of (a) product data, (b) No. of demands, and (c) resource flexibility

[1] Number of demands (ND): The number of demands increases exponentially in domain size of DFS problem. Four levels of the factor are set in the experiment as shown in Fig. 10b, which are ND=3 (nd03), ND=6 (nd06), ND=12 (nd12), and ND=60 (nd60). The demands in nd06, nd12, and nd60 are generated by splitting quantity of a demand in nd03 into 2, 4, and 20, respectively. The domain of nd03 is a subset of nd06, because requirements exploded from demands in nd03 can be composed by aggregating the requirements from demands in nd06. Similarly, nd06 and nd12 are sub-problems of

nd12 and nd60 respectively.

[2] Resource flexibility (RF): A resource is called flexible, if it works for more than one work centers. Resource flexibility is defined as the average number of work centers that a resource joins. Three levels of the factor, rf01, rf03, and rf05, are well prepared, as shown in Fig. 10c, to make the work center - resource pairs in rf01 and rf03 be subset of the pairs in rf03 and rf05, respectively.

[3] Evaluation function (EF): Three evaluation functions are performed in the experiment, which are makespan, service level, and tardiness.

There are 27 cases composed by 3 NDs, 3 RFs, and 3 EFs in the experiment, and each case runs 10 times. The result of each run is evaluated when the number of scheduling (SCH) equals 30000.

We run the cases on a laptop computer with Centrino Penitum® M 0.9 Ghz CPU, and the necessary times for running problem nd03, nd06, nd12, and nd60 are 50, 105, 250, and 2610 seconds, respectively. If the numbers of demands are 100, 200, and 300, the necessary times become 200, 1600, and 7500 minutes, respectively. The time needs to run a case grows up exponentially with the length of a chromosome.

		RF											
		rf01				rf03				rf05			
ND	EF	Best	Avg.	Diff.	CV	Best	Avg.	Diff.	CV	Best	Avg.	Diff.	CV
nd03	Makespan	14338	14338	0	0.000	11073	11254	181	0.013	11073	11352	279	0.023
	Service level	67	53	14	0.306	67	63	4	0.158	67	67	0	0.000
	Tardiness	9033	9164	131	0.036	4658	5712	54	0.097	4067	5381	1314	0.160
nd06	Makespan	13322	13333	11	0.008	10442	10569	127	0.009	10390	10465	75	0.005
	Service level	83	76	7	0.000	83	67	17	0.000	83	67	17	0.000
	Tardiness	6673	6976	303	0.020	1212	3420	2208	0.344	313	3713	3400	0.393
nd12	Makespan	13172	13194	22	0.002	10406	10601	195	0.003	10390	10628	238	0.004
	Service level	83	73	10	0.045	100	74	26	0.060	100	74	26	0.034
	Tardiness	6673	11667	4994	0.016	0	4246	4246	0.180	0	3379	3379	0.479
nd60	Makespan	13112	13587	475	0.003	10406	13498	3092	0.031	10390	11752	1362	0.007
	Service level	83	63	20	0.026	100	66	34	0.043	100	68	32	0.033
	Tardiness	6673	76189	69516	0.046	0	52360	52360	0.140	0	49856	49856	0.139

Table 3. The best and statistical results of the experiment

Table 3 shows the best value, average value, performance, and variance of the cases. The

'best' values, found so far, of the evaluation functions are discovered by setting parameters to keep MDGA in a divergence status for a long time. The average value is measured over 10 optimal values of a case. Performance of MDGA is defined as the difference between the average value and the best one. The variability of applying MDGA to runs of the cases in the experiment measured by coefficient of variance (CV), denoted by δ/μ, where δ is the standard deviation and μ the mean of the optimal values.

Since there is an inclusive relationship between levels of ND and RF, the smaller the lot size and the more flexible the resource, the better the best value can be found. However, increasing ND and RF not only provide MDGA with a better chance of optimization, but also enlarge domain size of the problem. The DFS problem with large domain size challenges the limits of MDGA's ability. As shown in the table, increasing ND and RF improves the optimal value at first, but it gets worse when ND and RF continue to increase.

The large ND aggravates the performance of MDGA. RF performs in a similar way with ND except that increasing RF won't delay the response time or severely worsen the performance of MDGA. A plant with high resource flexibility using MDGA against uncertainty is regarded as capable of responding to a change well and efficiently.

In general, the case setting makespan as evaluation function has low variability ($CV<0.1$). Whether or not service level performs stable depends very much on the problem. For some difficult cases like the combination of nd06 and rf03, the performance of service level obtained by setting tardiness as evaluation function is even better than by setting service level itself. There are n+1 degrees of service level if ND equals to n. Having few degree of evaluation makes MDGA easy to converge to a degree and dull to make a step toward a better degree. The cases setting tardiness as evaluation function has high variability ($CV>0.1$) when ND and RF are high. The solution to the high variability of tardiness is to run a case longer or set a larger population.

6. Conclusion

A genetic algorithm with MGG and demand crossover (MDGA) is proposed to solve dynamic flexible scheduling (DFS) problem. The problem is formulated and its domain is associated with the searching space of GA to encode a chromosome of MDGA. The problem is practical, goal-oriented, resource flexible, and capable of doing rescheduling dynamically. Though MDGA approach to DFS problem has its own value, this research is also an augmentation of agile production planning and control system (APPCS) that only generates a feasible schedule.

MDGA integrates minimal generation gap (MGG) and demand crossover. The effectiveness and correctness of MDGA have been shown by a benchmark and the exhaustive search. The formulation of DFS problem makes the exhaustive search possible.

The response time of MDGA to DFS problem increases exponentially with the length of a chromosome, which is determined by the shape of BOM, routing, and number of demands. Therefore, when MDGA is applied to a plant, to estimate execution time, it is necessary to calculate the length of a chromosome made from the BOM, routing, and demands. The experiment suggests that if the lengths of a chromosome are 700, 900, 2000, 3000, and 4500, then the response times will be 05 hour, 1 hour, 0.5 day, 1 day, and 5 days, respectively. A more efficient algorithm for a huge DFS problem will be a topic of future research.

A balance between the flexibilities and the ability of MDGA is a key point to get a better optimal value. The experiment for the example indicates that a double or triple flexibility improves about 10% - 25% of optimal value.

Forecasting is always wrong. Reserving safety buffers for a forecasting error is not the only way against unknown uncertainty. Forecasting revision is shown to be possible by APPCS and improved by MDGA.

Acknowledgement

One of the authors is grateful to the Research Assistant Grant of the University of Tsukuba, 2003, for partial support of the research.

Appendixes

Appendix A: The total number of legal permutations on requirements in *Rq*

The size of the set *SQ* depends on the size of the set *Rq*, and the shape of graph G_{rq} on *Rq*. An example of the graph is shown in Fig. 3. The graph is split into two branches b_1=⟨ rq_1, rq_2,…, rq_7 ⟩, b_2=⟨ rq_8, rq_9,…, rq_{14} ⟩, and both branches have two sub-branches b_{11}=⟨ rq_3, rq_4, rq_5 ⟩, b_{12}=⟨ rq_6, rq_7 ⟩, and b_{21}=⟨ rq_{10}, rq_{11}, rq_{12} ⟩, b_{22}=⟨ rq_{13}, rq_{14} ⟩, respectively.

The requirements in a branch is regulated by precedence constraints, but no such constraint exists among branches of the same level, e.g. b_{11} and b_{12}. A permutation on elements of the lower-level branches determines a sequence of the higher-level branch.

Let $N = \langle rq_i \rangle_{i=1}^{n}$ and $M = \langle rq_i \rangle_{i=1}^{m}$ be two legal sequences of requirements. A sequence $V = \langle rq_i \rangle_{i=1}^{n+m}$ is a legal permutation on $\{rq_i\}_{i=1}^{m} \cup \{rq_i\}_{i=1}^{n}$ if $V - \{rq_i\}_{i=1}^{m} = N$ and $V - \{rq_i\}_{i=1}^{n} = M$, where '–' is a function removing the elements in a set from a sequence without changing order of the sequence. The various requirements, whose precedence relation within *M* and *N* is unchanged, can be viewed as the same requirements in permutation. Hence, the total number of permutations is $(m+n)!/m!n!$ or C_m^{m+n}.

For example, $C_e^{3+2} = 10$ legal permutations of b_1 is determined by permutations on requirements in b_{11} and b_{12}. In a similar manner, b_2 also has 10 permutations. There are $C_7^{7+7} = 3{,}432$ legal permutations for a permutation of b_1 and a permutation of b_2. Size of the set of legal sequences $|SQ|$ in Fig. 4a is thus 10×10×3432=343200.

Appendix B: The number of requirement aggregations for a sequence of requirements

Assume there are *n* requirements with the same operation that are linked together

somewhere in a sequence. The n requirements can be put into 1, 2,…, m ($m \le n$) baskets with each basket having (c_1, c_2, …, c_m) requirements. For example, 6 requirements can be put into 3 baskets by ways of (4, 1, 1), (3, 2, 1), and (2, 2, 2). The number of alternatives to distribute n requirements into m baskets with each basket having (k_1, k_2, …, k_m) requirements is $(C_{k_1}^{n} C_{k_2}^{n-k_1} C_{k_3}^{n-k_1-k_2} ... C_{k_m}^{k_m}) / (b_1! b_2! .. b_v!)$, where b_i, i=1..v, are numbers of baskets whose number of requirements is equal and $\sum_{i=1}^{v} b_i = m$. For example, there are $(C_4^6 C_1^2 C_1^1) / (2! \times 1!) = 15$ ways to put 6 requirements into 3 baskets by way of (4, 1, 1). The numbers of ways for the other cases (3, 2, 1) and (2, 2, 2) are 10 and 15 respectively.

There are 7 groups of 2 requirements with the same operation linked together in the sequence sq_i shown in Fig. 4b. Each group has 2 ways of aggregation, i.e. either aggregate or not, hence $|QA_i| = 2^7 = 128$.

Appendix C: The number of WIP allocations

Let's first consider the problem of allocating q units of a WIP to n requirements with each requirement having c_i (i=1..n) units such that $\sum_{i=1}^{n} c_i = q$ and c_i is a natural number. This problem can be viewed as permutation of q units of WIP and n different requirements. Let 'o' represent a WIP, r_i (i=1..n) a requirement, and the permutation 'r_1 o o o r_5 r_3 o….' shows c_1=0, c_5=3, and c_3=0, i.e. the number of WIP before a requirement represents the allocated quantity. Since 'r_1 o o o r_5 r_3 o….' and 'r_5 o o o r_3 r_1 o….' represent the same set of WIP allocation, the precedence relation of requirements in a sequence must be fixed to avoid such duplication. Total number of permutations is thus $(q+n)!/q!n!$ or C_n^{q+n}.

Assume $|Wp|=u$, and (q_1, q_2,…, q_u) are the quantities of WIP allocated to number of requirements (n_1, n_2,…, n_u) in Rq, then there are $C_{n_1}^{q_1+n_1} \times C_{n_2}^{q_2+n_2} ... \times C_{n_u}^{q_u+n_u}$ ways of such allocation. The number of possible allocations for the case shown in Fig. 4d is $C_2^{8+2} = 45$.

References

Al-Hakim, L., 2001. An analogue genetic algorithm for solving job shop scheduling problems, International Journal of Production Research 39 (7) 1537-1548.

Aytug, H., Khouja, M., Vergara, F. E., 2003. Use of genetic algorithms to solve production and operations management problems: a review, International Journal of Production Research 41 (17) 3955-4009.

Croce, F. D., Tadei, R., Volta, G., 1995. A genetic algorithm for the job shop problem, Computers & Operations Research 22 (1) 15-24.

Hopp, W. J., Spearman, M. L., 2000. Factory Physics – Foundations of Manufacturing Management, 2nd edition, McGraw-Hill College.

Kacem, I., Hammadi, S., Borne, P., 2002. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems, IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews 32 (1) 1-13.

Nearchou, A. C., 2004. The effect of various operators on the genetic search for large scheduling problems, International Journal of Production Economics 88 (2) 191-203.

Sato, R., Tsai, T. L., 2004. An agile production planning and control with advance notification to change schedule, International Journal of Production Research 42 (2) 321-336.

Tsai, T. L., Sato, R., 2004. A UML model of agile production planning and control system, Computers in Industry 53 (2) 133-152.

Wang, L., Zheng, D., 2001. An effective hybrid optimization strategy for job-shop scheduling problems, Computers & Operations Research 28 (6) 585-596.

Yamamura, M., Satoh, H., Kobayashi, S., 1996. An analysis on generation alternation models by using the minimal deceptive problems, Journals of the Japanese Society for Artificial Intelligence 13 (5) 746-756. (in Japanese)