A rule-centric memetic algorithm to minimize the number of tardy jobs in the job shop

T. C. CHIANG† and L. C. FU‡*

† Full address:
Department of Computer Science and Information Engineering, National Taiwan University, No. 1, Sec. 4, Roosevelt Road, 106, Taipei, Taiwan, R.O.C.

Email: tcchiang@ieee.org

Telephone: (+886-2) 2362-2209

Fax: (+886-2) 2365-7887

‡ Full address:
Department of Computer Science and Information Engineering, National Taiwan University, No. 1, Sec. 4, Roosevelt Road, 106, Taipei, Taiwan, R.O.C.

Email: lichen@ntu.edu.tw

Telephone: (+886-2) 2362-2209

Fax: (+886-2) 2365-7887

* To whom correspondence should be addressed.

Abstract

This paper addresses the job shop scheduling problem with minimizing the number of tardy jobs as the objective. This problem is usually treated as a job sequencing problem, and the permutation-based representation of solutions was commonly used in the existing search-based approaches. In this paper, the flaw of the permutation-based representation is discussed, and a rule-centric concept is proposed to deal with it. A memetic algorithm is then developed to realize the proposed idea by tailored genome encoding/decoding schemes and local search procedure. Two benchmark approaches, a multi-start hill climbing approach and a simulated annealing approach, are compared in the experiments. The results show that the proposed approach significantly outperforms the benchmarks.

Keywords: job shop scheduling, memetic algorithm, priority rules

1. Introduction

Production scheduling is a research field that attracts researchers in the academia and engineers in the industry for last several decades. Many problems in this field fall into the category of discrete optimization problems and their NP-hard nature is challenging and interesting to researchers. In the manufacturing industry, scheduling is a critical issue in the phase of shop floor control. Engineers in the factory rely on a good scheduling approach to arrange the resources so that a satisfactory performance can be achieved. There are several well formulated problems in this field, such as single machine scheduling, flow shop scheduling, job shop scheduling, and so on. In this paper we take the job shop scheduling problem as the target since it is generic for the presence of multiple machines and allowance of different routes of jobs.

In the literature, the objectives of job shop scheduling problems can be divided into flow time-based ones and due date-based ones. The flow time of a job is the duration from when its first operation starts to when its last operation finishes. Among the flow time-based objectives, the makespan, which refers to the maximum flow time of all jobs, is the most well known one. Recently, due date-based objectives gradually received more attention from researchers since meeting due dates becomes an important requirement for industry practitioners. Given a due date, a job is marked as tardy if its last operation is finished later than the due date. In this paper, our objective is to minimize the number of tardy jobs, or to maximize the meet-due-date rate.

The paper proceeds as follows. Section 2 gives the problem definition of the job shop scheduling problem. In Section 3, we provide a literature survey for the approaches that were commonly used to solve the job shop scheduling problem. Section 4 describes the details of the proposed approach, which integrates two promising approaches, the priority rules and the memetic algorithms. Experiments and numerical results of the performance of the proposed approach and two benchmark approaches are given in Section 5. Finally, conclusions are made in Section 6.

2. Problem definition

Scheduling is actually a task of allocation of resources to requests under certain constraints to meet the concerned criteria. In a job shop, resources refer to the machines and requests are the jobs to be processed. Given N jobs and M machines in the job shop, each job i has M operations denoted by O_{ij}, $i = 1…N$, $j = 1…M$. An operation refers to processing of a job on a particular machine, and each job must be processed by each machine exactly once. Each operation O_{ij} has a fixed processing time p_{ij}. Let t_{ij} mean the starting time of operation O_{ij} in the schedule, its completion time C_{ij} can then be calculated as $C_{ij} = t_{ij} + p_{ij}$. For each operation O_{ij}, assume the preceding operation on its dedicated machine is O_{kl}. Then t_{ij} can not be smaller than $max\{C_{i(j-1)}, C_{kl}\}$ considering the precedence constraint of operations of a job and the machine capacity constraint. For any operation O_{ij} that is the first operation on its dedicated machine, the constraint of t_{ij} is modified to be no smaller than $C_{i(j-1)}$ only. The value of C_{i0} is zero for $i = 1…N$, which stands for that all jobs are ready to process at time zero. There are no setup times, no preemption, and no machine breakdowns. The number of buffers is assumed infinite, and the transportation issue is not considered. Each job i has a due date d_i. We set U_i as one if C_{iM} is greater than d_i; otherwise, U_i is set as zero. The objective in this paper is to find a schedule that minimizes the number of tardy jobs $\sum_{i=1...N} U_i$, or equivalently, to maximize the meet-due-date rate $1 - \sum_{i=1...N} U_i$ /N.

3. Literature survey

The job shop scheduling problem is essentially a job sequencing problem – to determine the processing orders of jobs on each machine to satisfy the performance criterion. Among the approaches to this sequencing problem, priority rules are pervasively adopted in the industry due to its ease of implementation, small computation requirement, satisfactory performance, and the flexibility to incorporate domain knowledge and engineers' expertise. The wide acceptance can be seen in Appleton-Day and Shao (1997), Giegandt and Nicholson (1998), Hsu and Lan (2000), and Kim *et al.* (2001). By using the priority rule, each time a machine is free, the rule is invoked to calculate a priority value for each waiting job in the queue of this machine. Then the job with the highest priority value is

taken as the next one to process. To develop a rule, researchers need to identify the important factors relevant to jobs and machines, and then manipulate these factors to come up with an appropriate numerical value as the priority. Vepsalainen and Morton (1987) proposed a parameterized rule, well known as the Apparent Tardiness Cost (ATC) rule to minimize weighted tardiness in the job shop. It assigned priorities to jobs according to the expected delay cost per imminent machine processing time. Anderson and Nyirenda (1990) developed two rules using dynamic operation due dates based on the remaining allowance times to minimize due date-based objectives in the job shop. The shop floor utilization level was taken to adjust the weights to the processing time and due date information in the rule proposed by Raghu and Rajendran (1993). A critical ratio-based rule exploiting group information of jobs was devised by Chiang and Fu (2004) to minimize the number of tardy jobs. For other reports and surveys of priority rules, see Panwalkar (1977), Kim and Kim (1994), Chang *et al*. (1996), Sabuncuoglu (1998), and Costa and Ferreira (1999).

Local search-based approaches were also common for solving job sequencing problems since they are general and often provide promising results. To use local search-based approaches, a complete schedule should be given as an initial solution. Then a search process proceeds by examining the neighboring solutions generated from the so-called neighborhood functions. Different searching strategies in the neighborhood result in different local search approaches, such as simulated annealing (SA)(Van Laarhoven *et al*. 1987) and tabu search (Glover 1989). He *et al*. (1996) showed a SA approach for due-dates job shop scheduling. Armentano and Ronconi (1999) and Yang *et al*. (2004) adopted the tabu search approach to minimize total tardiness in the flowshop. Combining local search-based approaches is also a possible alternative, as can be seen in Adenso-Diaz (1996) and Kreipl (2000). Other surveys of local search-based approaches can be found in Crauwels *et al*. (1996), Dorn *et al*. (1996), and Vaessens *et al*. (1996).

After Goldberg's book was published in 1989, the genetic algorithm became popular in solving optimization problems. Unlike local search-based approaches, the genetic algorithm is a global search approach achieved by a population-based search process. It mimics the evolutionary process in the nature by realizing Darwin's principle – natural selection and survival of the fittest with artificial genetic

operators including selection, crossover, mutation, and so on. Solutions are first encoded as genomes and constitute a population. Based on the performance/quality of the corresponding solution, each genome is given a fitness value. Then genomes are selected according to their fitness values so that fitter genomes participate more times in offspring breeding. When parents are selected, the offspring are generated by crossover and mutation. At last, the reproduction scheme decides which genomes can survive to the next generation. Generation by generation, genomes become better and better, and finally we can expect that an optimal or near-optimal solution will be obtained. Different encoding/decoding mechanisms and different implementations of the genetic operators make up various genetic algorithms. A comprehensive survey of job shop scheduling using genetic algorithms was provided by Cheng *et al*. (1996, 1999).

Although the genetic algorithm has the global search ability, sometimes it was outperformed by other search algorithms due to its slow convergence speed (Vaessens *et al*. 1996, Dorn *et al*. 1996, Mattfeld and Bierwirth 2004, Goncalves *et al*. 2005). To increase the performance of genetic algorithms, a new approach called the memetic algorithm (Moscato 1989), was growing in recent years. The memetic algorithm, also known as the genetic local search, is a combination of the genetic algorithm and the local search-based approach in order to possess both the global search ability and search efficiency from these two kinds of approaches. Cai *et al*. (2000) proposed a memetic algorithm which embedded a hill climbing local search procedure to minimize the makespan in the job shop. The SA algorithm was adopted as the local search component to minimize the makespan in the job shop in Wang and Zheng (2001). In Goncalves *et al*. (2005), the hill climbing was also used but with a different neighborhood function from Cai *et al*. (2000). Other applications of the memetic algorithm on job sequencing problems can be found in Murata *et al*. (1998), Franca *et al*. (2001), and Sevaux and Dauzere-Peres (2003).

4. The rule-centric memetic algorithm

As what we can see from the literature, priority rules and the memetic algorithms are two promising approaches to solve the job shop scheduling problem. Priority rules break the job shop scheduling problem into many decisions of processing orders of jobs, and make the decisions dynamically based on the status

of the shop. The memetic algorithms, on the other hand, intend to solve the job shop scheduling problem by searching among a very large number of possible solutions efficiently through a sophisticated search mechanism. In this work, we will propose an approach which exploits the dynamic decision ability of priority rules and the search ability of memetic algorithms in an integrated manner. The details will be given in this section. Before going into the details of each component in our approach, we give an overview of the entire procedure first as follows:

Step 0. Determine the genome encoding and decoding schemes. (Section 4.2, 4.3)

Step 1. Initialize the population, and evaluate their fitness. (Section 4.8)

Step 2. Reproduce the population in the next generation.

Do the following steps with a certain times.

Step 2.1 Apply selection to pick two individuals as parents (Section 4.5)

Step 2.2 Apply crossover to generate two offspring. (Section 4.5)

Step 2.3 Apply mutation to each offspring probabilistically. (Section 4.5)

Step 2.4 Determine which two individuals can survive based on the reproduction scheme. (Section 4.6)

Step 3. Apply the local search procedure to the population. (Section 4.7)

Step 4. Evaluate the individuals in the population. (Section 4.4)

Step 5. If the stopping criterion is reached, ends; otherwise, go back to Step 2.

4.1 The concept of rule-centric

As mentioned, to solve the job shop scheduling problem is to sequence the operations on the machines. In the literature, the solution was usually encoded as a permutation of operations. Then the corresponding schedule is derived by processing operations on each machine following the order in the permutation. Figure 1 gives an example of this conventional encoding and decoding mechanism.

[Insert figure 1 about here]

With the permutation-based representation, new solutions are usually generated by changing the order of operations, for example, by pairwise interchange (also known as swap) and insertion (also known as shift) in the local search-based approaches, as illustrated in figure 2. In this way, different schedules can be derived from different permutations of operations.

[Insert figure 2 about here]

However, we think this kind of local and static modification could have some problem that reduces the search efficiency. We use figure 3 to explain the potential problem. We have four jobs and two machines, and the permutation which indicates the processing orders of operations on the machines is as illustrated in the figure. Suppose job 3 is tardy and the insertion (INS) neighborhood function is used. After applying INS, the new permutation is formed by moving the operation O_{31} to be the first to be processed on machine M1 while the relative processing order of all other operations are retained. The problem is that should the processing order of the remaining operations on M1 change adaptively? For example, if the operation O_{11} becomes tardy inevitably after processing O_{31} first, O_{11} should not be processed earlier than the other two operations. Besides the need of adaptively changing the processing order of the remaining operations on the same machine as O_{31}, we may also need to adjust the order of operations on the machine (M2) on which the succeeding operation of O_{31} (namely, O_{32}) is to be processed. In the original permutation, the operation O_{32} will be processed at the last on M2. If this order is not changed in the new permutation, then processing O_{31} at the first on M1 is useless to make job 3 meet its due date since the finish time of this job is not getting earlier.

[Insert figure 3 about here]

In short, we hope that each time we make a modification in one part of the schedule, a chain reaction will take place to broadcast this modification and the entire schedule is adjusted accordingly. This requirement reminds us of the dynamic decision ability of priority rules. In practice, a priority rule is applied to determine the processing order of operations based on the shop status when a machine is

available. This characteristic will be utilized in our approach to realize the chain reaction. Note that the integration scheme proposed here is not the same as the conventional one to combine priority rules and the search mechanism. In the conventional scheme, the priority rule is simply used to generate the initial solution(s) for the search mechanism. However, in the proposed rule-centric scheme, the priority rule and the search mechanism will interact with each other. Figure 4 depicts the relationship between the priority rule and the search mechanism in the conventional and the proposed integration scheme. Details of the interaction between the priority rule and the search mechanism will be explained in the following sub-sections.

[Insert figure 4 about here]

4.2 Genome encoding

Since we use the priority rule to dynamically determine the processing order of operations, we do not record the order of operations statically as the permutation-based representation does. Instead, each operation is associated with a level, and the genome in our memetic algorithm is a string of these levels of all operations. The spirit of the proposed rule-centric approach is that the adopted priority rule is expected to generate a correct processing order for most operations, and the levels associated with operations are used only if some correction is necessary. The level is an integer in current implementation, and an example of the genome is given in figure 5 (a).

[Insert figure 5 about here]

4.3 Genome decoding

Given a genome, a schedule is derived based on the associated levels of operations recorded in the genome and the adopted priority rule. Each time when a machine is available, the operations waiting to be processed are firstly collected. In figure 5 (b), for example, we have three awaiting operations – O_{11}, O_{21}, and O_{31}. Then the operations with the highest level are ranked by the priority rule to decide the next processing target. From the genome in figure 5 (a), the levels of operations O_{11} and O_{12} are one, and the level of operation O_{31} is zero. Hence, only operations O_{11} and O_{12} will be considered by the priority rule.

At last, the machine starts to process the selected target. This selection process will take place each time when a machine becomes available. Here, when machine M1 finishes O_{11}, the priority rule will be invoked again to determine the next processing target.

[Insert figure 6 about here]

Assume that the level of operation O_{31} is changed to two, as illustrated in figure 6 (a), then another schedule will be derived. In figure 6 (b), only O_{31} has the highest level (two) among three awaiting operations. Therefore, O_{31} is certainly the next processing target. Comparing to figure 5 (b), the first operation processed on machine M1 is changed from O_{11} to O_{31}. The shop status after machine M1 processed the first operation in figure 5 (b) and 6 (b) might be different, and the second-time invocation of the priority rule in these two cases might make different decisions. This is the dynamic decision ability that we mentioned in section 4.1. A change of the first operation can cause adaptive changes of processing orders of other operations.

In addition to the influence on the candidate operations of the priority rule, the associated levels of operations are also used for the virtual preemption during construction of the schedule. When a machine is busy and an operation with a higher priority level arrives, the in-process operation will be preempted. The entire process of the preempted operation needs to restart (not resume) when next time it is selected to be processed. “Virtual” refers to the fact that the preemption only takes place during schedule generation and does not really happen in reality. This virtual preemption paradigm was proposed in Chiang *et al*. (2005), and was shown to improve the performance of the conventional paradigm of using priority rules.

[Insert figure 7 about here]

4.4 Fitness function

In our memetic algorithm, the fitness value of each genome is calculated according to two performance measures of the corresponding schedule. The primary measure is the meet-due-date rate, which is the objective in our target problem. Since the benefit of incorporating the second criterion was presented in

Duvivier *et al*. (1998) and Hertz and Widmer (2003), we add the total tardiness as the second measure to help identifying the quality of genomes in a finer manner. Denote the meet-due-date rate and total tardiness of a genome g by r_g and t_g, the fitness value f_g is calculated by the following equation:

$$f_g = w_1 \cdot (r_g - \min\{r_i\})/(\max\{r_i\} - \min\{r_i\}) + w_2 \cdot (\max\{t_i\} - t_g)/(\max\{t_i\} - \min\{t_i\}) \; i \in \text{current population}$$

Simply speaking, the fitness value is a linear weighted sum of the normalized values of the meet-due-date rate and total tardiness. The values of weights will be set through preliminary experiments.

4.5 Selection, crossover, and mutation

Roulette wheel selection is adopted, in which a genome is selected as a parent in the probability in proportion to its fitness value. Since we do not encode the processing order of operations explicitly in the genome, simple crossover operators can be applied. Here we tested one-point crossover and two-point crossover. By a preliminary experiment, two-point crossover gave better performance and was chosen. As for mutation, we randomly pick an operation on each machine, and its associated level is increased or decreased by one in equal probability.

4.6 Reproduction scheme

The reproduction scheme determines which individuals in the current generation can survive to the next generation. Besides the biased selection operators like roulette wheel selection, the reproduction scheme is the main force to push the entire population to converge to better individuals. We tested three existing schemes in our approach, but found that their performance is not good in our application.

The first scheme was proposed in Goncalves *et al*. (2005), and is illustrated in figure 8 (a). In this scheme, the best several individuals in the current generation are first copied to the next generation. Then a portion of the population in the next generation is filled with offspring produced by mating the individuals in the current population. Finally, some randomly generated individuals are included to complete the entire population. In our test, this scheme did not provide good convergence pressure due to two reasons: first, the offspring produced by mating are directly put into the next generation but they do not guarantee to be

better than their parents; second, individuals brought in by random immigration are usually much worse than other individuals after the evolutionary process proceeds several generations, and hence this portion of population has little effects.

In order to strengthen the convergence pressure, we then tested the second scheme, which was used in Wang and Zheng (2001). In this scheme, n offspring are produced by mating where n is the population size. Then best n individuals from the current population and offspring survive to the next generation. This $n/2n$ scheme is illustrated in figure 8 (b). Strong convergence pressure was exhibited in the test, but it was also the problem. When two parents with high fitness produce two offspring that also have high fitness, all four individuals may survive and which means the size of this family doubles. This phenomenon causes premature convergence in only several generations and reduces the searching efficiency.

The third scheme, which can be seen as a modified version of the second one, is shown in figure 8 (c). In this scheme, best two individuals of the two selected parents and two offspring produced by mating survive to the next generation. This is called the 2/4 scheme, and was adopted in Cai *et al*. (2000). In our test, the premature convergence phenomenon still appeared, though later than in the second scheme.

After trying three existing schemes, we found that they are not suitable in our application. Therefore, a new scheme is proposed here. The major difference between our scheme and the third scheme is that the best two individuals among two parents and two offspring "replace" the parents so that we do not have a fast growth of a family. To maintain diversity, we further restrict that the best two individuals must have different meet-due-date rates or total tardiness. (The only exception is when all four individuals have the same meet-due-date rate and total tardiness.) Random immigration in the first scheme is also employed here, but they will replace the worst individuals in our scheme, which is different from the way in which the first scheme does.

[Insert figure 8 about here]

4.7 Local search procedure

'GA can easily identify different solution subspaces with good characteristics, but they lack the "killer instinct" that would allow them to intensify the search in these areas (Taillard *et al*. 2001).' To provide the fine-grained search ability, the local search procedures are incorporated into the genetic algorithms, and applications of the resulted memetic algorithm grew in recent years. There are three key components in the design of a local search procedure – configuration representation, neighborhood function, and searching strategy. Here the representation of configuration is the same as that of genome, and the details of other two components are given in the following subsections.

4.7.1 Probabilistic level up (PLU) neighborhood

In the proposed approach, we expect that the adopted priority rule should determine appropriate processing order of most operations. Certainly the rule might not generate a perfect order and some operations that should be processed early are processed late. We push these delayed operations to be processed earlier by increasing their associated levels. This is the main idea of our neighborhood function.

To generate a neighboring genome g_n of a base genome g_b, firstly the tardy jobs in the corresponding schedule of g_b are identified. Then each of these jobs is marked as 'to-be-leveled-up' in probability P_J. Finally, for each of these 'to-be-leveled-up' jobs, each of its operations that experience positive waiting time receives an increment of level by one in probability P_O. Let us use figure 9 to explain how it works. Assume that job 2 and job 3 are tardy in the corresponding schedule of g_b. Assume $P_J = 0.5$, which means that on average half of the tardy jobs will be marked as 'to-be-leveled-up,' and $P_O = 0.5$, which means on average half of operations will receive an increment of level. Among job 2 and 3, suppose job 3 is marked as 'to-be-leveled-up.' Then among two operations of job 3, suppose O_{31} is selected to receive an increment of level. Finally we can get the neighboring genome as illustrated. We call this neighborhood function 'probabilistic level up (PLU),' and the idea behind it is to try to make a portion of tardy jobs meet due dates by leveling up some operations of these jobs so that these operations are processed earlier and consequently push these jobs to be finished earlier.

[Insert figure 9 about here]

4.7.2 Searching strategy

The configuration determines the search space; the neighborhood function and the evaluation function sketch the landscape of the search space, and the searching strategy leads the path during searching. Among several searching strategies, the hill climbing strategy is the most common in the local search procedure in memetic algorithms because it provides intensive search ability and does not take too much computation time (Goncalves *et al.* 2005, Sevaux *et al.* 2003). We also use this strategy in our local search procedure. When a neighboring genome is generated, it is accepted only if it has higher meet-due-date rate, or the same meet-due-date rate and lower total tardiness. For each base genome, a fixed number of neighboring genomes are generated. If no genome in the neighborhood is better, the local search procedure stops.

In section 4.7.1, we mentioned that there is a parameter P_J in the neighborhood function. This parameter represents the percentage of jobs to be 'saved' from being tardy. To apply the local search procedure to a genome g, the value of P_J is set as $P_J^0/(1+l_g)$, where P_J^0 is the parameter of the memetic algorithm, and l_g is the number of applications of the local search procedure to this genome. This adjustment reflects the idea that the percentage of jobs that can be 'saved' would get smaller and smaller as the genome is improved by the local search procedure more and more times. By decreasing P_J based on the search progress, the resulted neighborhood by the PLU neighborhood function will also change adaptively. It would be beneficial to generate good solutions more easily.

4.8 Generation of the initial population

In the initial population, there is always one genome with all zero levels. This genome represents the schedule purely determined by the adopted priority rule. The remaining population is filled with genomes generated by randomly assigning each level with zero and one in equal probability.

5. Experiments and results

5.1 Generation of problem instances

We generated two data sets, each with ten instances. There are 20 machines. The processing time of each operation is a random number in [1, 50], and setup time is considered to be included in the processing time. The numbers of jobs are 20 in the first data set and 60 in the second one, respectively. Each instance is generated by determining the processing route of each job and the processing time of each operation randomly. The due date of each job is assigned by $r \cdot p$, where r is a random real number in [1, d] and p is the total processing time of this job. The value of d is chosen so that the meet-due-date rate of each problem instance is approximately between 0.75 and 0.8 when the ECR rule (Chiang and Fu 2004) is used.

5.2 Two benchmark approaches

To examine the performance of the proposed memetic algorithm, two other approaches are implemented. They are local search-based approaches, and use the conventional permutation-based representation. For each approach, both pairwise interchange and insertion neighborhood functions are considered. These two neighborhood functions are pervasively used with the permutation-based representation, for example, see Cai *et al.* (2000), Hsieh *et al.* (2003), and Yang *et al.* (2004).

The first approach is a multi-start hill climbing (also known as descent search) approach. The initial solution is generated by a selected priority rule. Then a hill climbing search starts. For a base configuration, a fixed number of neighboring solutions are generated. Whenever no better solution is found in the neighborhood, the search process restarts again from the initial solution. The above process will repeat until the time limit is reached. At last, the best solution found during searching is reported. The multi-start hill climbing is a common benchmark, as can be seen in Crauwels *et al.* (1996), Franca *et al.* (2001), and Rajendran and Ziegler (2003). In the experiments, we set the time limit as four minutes.

The second approach is a SA algorithm. In SA, better configuration in the neighborhood is absolutely accepted. Besides, worse configuration can also be accepted, but with a probability related to the loss of solution quality. This kind of strategy is useful to jump out from the local optima and to reach the global optima. In our implementation, a worse neighboring configuration is accepted with the probability as

follows:

c_b: the base configuration, c_n: the neighboring configuration,

$r(.)$: meet-due-date rate, $t(.)$: total tardiness, T: temperature, $exp(.)$: exponential distribution

Case 1: If $r(c_n) < r(c_b)$, prob. = $exp(-(1 - r(c_n)/r(c_b))/T)$

Case 2: If $r(c_n) = r(c_b)$ and $t(c_n) > t(c_b)$, prob. = $exp(-(t(c_n)/t(c_b) - 1)/T)$

The cooling schedule has three parameters, the initial temperature, the final temperature, and the cooling rate. The initial temperature is 0.36, obtained by accepting a neighboring configuration with a 50% loss of quality in probability 0.25 (i.e. $exp(– 0.5/0.36) = 0.25$). The final temperature is 0.0027, obtained by accepting a neighboring configuration with a 1% loss of quality in probability 0.025. The cooling rate is set to control the computation time to be four minutes.

5.3 Experimental results

In our first experiment, the performance of the multi-start hill climbing approach was examined. We used five priority rules to generate the initial configuration – ECR (Chiang and Fu 2004), modified due date (MDD), shortest processing time (SPT), least slack (SLACK), and random (RND) rule. The equations for the first four rules are given in Appendix. Three neighborhood sizes, 20, 40, and 60, were tested. Hence, totally we had thirty versions of this multi-start hill climbing approach by five rules, three neighborhood sizes, and two neighborhood functions (pairwise interchange and insertion). Each problem instance was solved by each version by five times, and the minimal, average, and maximal meet-due-date rates among five runs were recorded. Then the averages of the minimal, average, and maximal meet-due-date rates over ten problem instances in each of two data sets were calculated. These three values were taken to measure the performance of each version in the worst, average, and the best cases, respectively. Since the results obtained by using pairwise interchange and insertion as neighborhood functions were quite similar, we only show the results by using pairwise interchange in table 1.

[Insert table 1 about here]

In table 1, we first observe that the initial solution has a significant impact on the performance. Using the ECR rule to generate the initial solution can provide the best performance. The MDD and SLACK rules have close performance, and the SPT rule is worse than the previous three rules. In data set 1, using the RND rule to generate the initial solution is better than using the SPT rule, however, the result is reversed in data set 2. It reveals that with larger problem size, which implies a larger solution space, the chance for a random initial solution to fall in a good region becomes smaller. The benefit of the domain knowledge in priority rules shows up more evidently.

As for the effects of the neighborhood size, larger neighborhood size is useful for all rules except the RND rule in data set 1. A large neighborhood size could cause the search process to take much time in a region even if the initial solution generated by the RND rule is bad. Therefore, the RND rule prefers a smaller neighborhood because it can try more initial solutions to find a good region within the computation time limit. In data set 2, the RND still performs better with a smaller neighborhood. For the other rules, the effect of the neighborhood size is not obvious.

In the second experiment, we studied the performance of the SA benchmark approach. Like the first experiment, we also tested thirty versions of this approach. The versions with insertion as the neighborhood function performed slightly better than those with pairwise interchange. Hence, we report the results of the versions using insertion neighborhood function in table 2.

[Insert table 2 about here]

In data set 1, we find that the impact of the initial solution reduces with the global search ability of the SA approach. No matter which rule is used to generate the initial solution, we can obtain close performance in the best case. Nevertheless, the initial solution still has a great influence on the performance in data set 2. The reason could be that the computation time becomes insufficient for the SA approach to reach a stable state at each temperature when the problem size gets larger.

Performance of the proposed memetic algorithm was investigated in the third experiment. Since the priority rule is expected to generate an appropriate processing order of operations in our approach, the

RND rule was not considered. In order to prevent the local search procedure from taking too much computation time, we tested three smaller neighborhood sizes, 10, 20, and 30. The population size was 20, and the mutation rate was 0.2. The weights w_1 and w_2 in the fitness function were set as three and one, which were obtained from a preliminary test on several different combinations of values. In each generation, 16 offspring are generated, and the worst four individuals are replaced by random immigration. The value of P_J^0 mentioned in section 4.7.2 is 0.5. The time limit is also four minutes, which is the same as the two benchmarks. There are twelve versions of the proposed approach in total by four priority rules and three neighborhood sizes, and the results are shown in table 3.

[Insert table 3 about here]

The performance obtained by adopting different rules is different, especially in data set 2, whose instances have larger problem size. This can be easily realized since the priority rule plays an important role in our "rule-centric" approach. The proposed approach performs the best with the ECR rule. The MDD and SLACK rules follow, and the SPT rule is the worst again. To solve problem instances in data set 2, the adopted rule must manage more operations than it does to solve instances in data set 1. Hence, the dynamic decision ability of different rules leads to much more significant performance gap.

In the proposed approach and two benchmarks, each priority rule was tested with many different settings (such as different neighborhood sizes). The best results obtained among those settings are now summarized in table 4. The first four rows show the meet-due-date rates obtained by purely using the priority rules. Table 4 shows that no matter which rule is adopted, the proposed rule-centric memetic algorithm significantly outperforms the two benchmark approaches in either the best, average, or worst cases. The meet-due-date rate is raised by 5.5% at least and 9.5% at most in data set 1. In data set 2, the increment is between 5.5% and 7.8%. These are significant improvement.

[Insert table 4 about here]

6. Conclusions

In recent years, search-based approaches are common to solve job shop scheduling problems. Whether local search-based or population-based search methods are used, the representation of solution is undoubtedly the most important issue. In this paper, first we show the flaw of the permutation-based representation, which is a popular representation in the literature. To eliminate this flaw, we proposed an idea that a modification in one part of the schedule must trigger a chain reaction to broadcast this modification and the entire schedule is adjusted accordingly. The dynamic decision ability of priority rules is suitable to realize this idea, and thus we developed a rule-centric memetic algorithm that tightly coupled the search process and the priority rules. New genome encoding/decoding schemes, reproduction schemes, and local search procedure were devised. The performance of our approach was shown evidently superior to two search-based approaches in the experiments.

Incorporation of domain knowledge and heuristics in the search-based algorithms is an unavoidable trend, as reported in Cheng *et al.* (1999) and Taillard *et al.* (2001). The rule-centric concept in our approach is actually one kind of realization. Domain knowledge and heuristics are incorporated into the memetic algorithm in the form of priority rules. The priority rule constructs the schedule based on the embedded knowledge; the search mechanism makes some corrections to the schedule; and then the priority rule adapts the schedule to the corrections by its dynamic decision ability. Through the interactions between priority rules and the memetic search mechanism, schedules with higher quality can be obtained more easily, as can be seen in the experiments. There is no restriction on the priority rules which can be used in the proposed approach. Managers and engineers in the shop can integrate their rules into this approach effortlessly. Domain knowledge could also be added in other components in the proposed approach, for example, the crossover, mutation, and the neighborhood function. This will be the research direction in our future work.

Appendix

Four priority rules used in this work

Rules	Priority value	Notations
SPT	$Z_i = p_i$	p: processing time of the imminent operation
MDD	$Z_i = \max\{d_i, t + r_i\}$	r: remaining processing time of the job (including p)
SLACK	$Z_i = d_i - t_i - r_i$	t: system time, the time at which the dispatching decision is to be made
ECR	$Z_i = \sum_{j \in competing\ jobs} (\frac{r_j}{d_j - t - p_j})^2 + (\frac{r_i - p_i}{d_i - t - p_i})^2$	d: due date of the job

References

Adenso-Diaz, B., 1996, An SA/TS mixture algorithm for the scheduling tardiness problem, *European Journal of Operational Research*, **88**, 516 – 524.

Appleton-Day, K. and Shao, L., 1997, Real-time dispatch gets real-time results in AMD's Fab25, *Proc. of IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop*, 444 – 447.

Anderson, E. J. and Nyirenda, J. C., 1990, Two new rules to minimize tardiness in a job shop, *International Journal of Production Research*, **28**, 2277 – 2292.

Armentano, V. A. and Ronconi, D. P., 1999, Tabu search for total tardiness minimization in flowshop scheduling problems, *Computers and Operations Research*, **26**, 219 – 235.

Cai, L.W., Wu, Q.H., and Yong, Z.Z., 2000, A genetic algorithm with local search for solving job problems, *Lecture Notes on Computer Science*, **1803**, 107 – 116.

Chang, Y.-L., Sueyoshi, T., and Sullivan, R.S., 1996, Ranking dispatching rules by data envelopment analysis in a job shop environment, *IIE Transactions*, **28**, 631 – 642.

Cheng, R., Gen, M., and Tsujimura, Y., 1996, A tutorial survey of job-shop scheduling problems using genetic algorithms – I. representation, *Computers in Engineering*, **30**, 983 – 997.

Cheng, R., Gen, M., and Tsujimura, Y., 1999, A tutorial survey of job-shop scheduling using genetic algorithms – II. Hybrid genetic search strategies, *Computers in Engineering*, **36**, 343 – 364.

Chiang, T.-C. and Fu, L.-C., 2004, Solving the FMS scheduling problem by critical ratio-based heuristics and the genetic algorithm, *Proc. of IEEE Conference on Robotics and Automation*, 3131 – 3136.

Chiang, T.-C. and Fu, L.-C., 2005, A virtual preemption paradigm for using priority rules for solve job shop scheduling problems, *Proc. of IEEE Conference on Robotics and Automation*, 3714 – 3719.

Costa, M.T. and Ferreira, J.S., 1999, A simulation analysis of sequencing rules in a flexible flowline, *European Journal of Operational Research*, **119**, 440 – 450.

Crauwels, H.A.J., Potts, C.N., and Van Wassehnhove, L.N., 1996, Local search heuristics for single-machine scheduling with batching to minimize the number of late jobs, *European Journal of Operational Research*, **90**, 200 – 213.

Croce, F.D., 1995, Generalized pairwise interchanges and machine scheduling, *European Journal of Operational Research*, **83**, 310 – 319.

Dorn, J., Girsch, M., Skele, G., and Slany, W., 1996, Comparison of iterative improvement techniques for schedule optimization, *European Journal of Operational Research*, **94**, 349 – 361.

Duvivier, D., Preux, Ph., Fonlupt, C., Robilliard, D., and Talbi, E.-G., 1998, The fitness function and its impact on local search methods, *Proc. of IEEE International Conference on Systems, Man, and Cybernetics*, **3**, 11 – 14.

Franca, P.M., Mendes, A., and Moscato, P., 2001, A memetic algorithm for the total tardiness single machine scheduling problem, *European Journal of Operational Research*, **132**, 224 – 242.

Giegandt, A. and Nicholson, G., 1998, Better dispatch application – a success story, *Proc. of IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop*, 396 – 399.

Glover, F., 1989, Tabu search-part I, *ORSA Journal on Computing*, **1**, 190 – 206.

Glover, F., 1989, Tabu search-part II, *ORSA Journal on Computing*, **2**, 4 – 32.

Goldberg, D.E., 1989, *Genetic algorithms in search, optimization and machine learning*, MA: Addison-Wesley.

Goncalves, J.F., Mendes, J.J. de M., and Resende, M.G.C., 2005, A hybrid genetic algorithm for the job shop scheduling problem, *European Journal of Operational Research*, **167**, 77 – 95.

He, Z., Yang, T., and Tiger, A., 1996, An exchange heuristic imbedded with simulated annealing for due-dates job-shop scheduling, *European Journal of Operational Research*, **91**, 99 – 117.

Hertz, A. and Widmer, M., 2003, Guidelines for the use of meta-heuristics in combinatorial optimization, *European Journal of Operational Research*, **151**, 247 – 252.

Hsieh, J.-C., Chang, P.-C., and Hsu, L.-C., 2003, Scheduling of drilling operations in printed circuit board factory, *Computers & Industrial Engineering*, **44**, 461 – 473.

Hsu, K.H. and Lan, C.C., 2000, Photolithography area dispatching scheme for advanced technology in foundry fabs, *Proc. of Semiconductor Manufacturing Technology Workshop*, 211 – 216.

Kim, M.H. and Kim, Y.-D., 1994, Simulation-based real-time scheduling in a flexible manufacturing system, *Journal of Manufacturing Systems*, **13**, 85 – 93.

Kim, Y.-D., Kim, J.G., Choi, B., and Kim, H.-U., 2001, Production scheduling in a semiconductor wafer fabrication facility producing multiple product types with distinct due dates, *IEEE Trans. on Robotics and Automation*, **17**, 589 – 598.

Kreipl, S., 2000, A large step random walk for minimizing total weighted tardiness in a job shop, *Journal of Scheduling*, **3**, 125 – 138.

Mattfeld, D.C. and Bierwirth, C., 2004, An efficient genetic algorithm for job shop scheduling with tardiness objectives, *European Journal of Operational Research*, **155**, 616 – 630.

Moscato, P., 1989, On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms, Caltech Concurrent Computation Program, C3P Report 826.

Murata, T., Ishibuchi, H., and Gen, M., 1998, Neighborhood structures for genetic local search algorithms, *Proc. of International Conference on Knowledge-based Intelligent Electronic Systems*, 259 – 263.

Panwalkar, S.S., 1977, A survey of scheduling rules, *Operations Research*, **25**, 45 – 61.

Raghu, T. S. and Rajendran, C., 1993, An efficient dynamic dispatching rule for scheduling in a job shop, *International Journal of Production Economics*, **32**, 30 – 313.

Rajendran, C.and Ziegler, H., 2003, Scheduling to minimize the sum of weighted flowtime and weighted tardiness of jobs in a flowshop with sequence-dependent setup times, *European Journal of Operational Research*, **149**, 513 – 522.

Sabuncuoglu, I., 1998, A study of scheduling rules of flexible manufacturing systems: a simulation

approach, *International Journal of Production Research*, **36**, 527 – 546.

Sevaux, M. and Dauzere-Peres, S., 2003, Genetic algorithms to minimize the weighted number of late jobs on a single machine, *European Journal of Operational Research*, **151**, 296 – 306.

Taillard, E.D., Gambardella, L.M., Gendreau, M., and Potvin, J.-Y., 2001, Adaptive memory programming: a unified view of metaheuristics," *European Journal of Operational Research*, **135**, 1 – 16.

Vaessens, R.J.M., Aarts, E.H.L. and Lenstra, J.K., 1996, Job shop scheduling by local search, *INFORMS Journal on Computing*, **8**, 302 – 317.

Van Laarhoven, P.J.M. and Aarts, E.H.L., 1987, *Simulated annealing: theory and applications*, Reidel, Dordrecht.

Vepsalainen, A.P.J. and Morton, T.E., 1987, Priority rules for job shops with weighted tardiness costs, *Management Science*, **33**, 1035 – 1047.

Wang, L. and Zheng, D.-Z., 2001, An effective hybrid optimization strategy for job-shop scheduling problems, *Computers & Operations Research*, **28**, 585 – 596.

Yang, T., Kuo, Y., and Chang, I., 2004, Tabu-search simulation optimization approach for flow-shop scheduling with multiple processors – a case study, *International Journal of Production Research*, **42**, 4015 – 4030.

Table 1. Performance of the multi-start hill climbing approach

		Data set 1 (20 jobs)			Data set 2 (60 jobs)		
Rule	Nb size	Min.	Avg.	Max.	Min.	Avg.	Max.
ECR	20	0.790	0.798	0.800	0.817	0.819	0.823
	40	0.795	0.806	0.815	0.815	0.820	0.825
	60	0.810	0.814	0.820	0.812	0.819	0.825
MDD	20	0.735	0.748	0.760	0.675	0.679	0.685
	40	0.745	0.760	0.775	0.675	0.681	0.690
	60	0.750	0.766	0.780	0.675	0.683	0.692
SPT	20	0.700	0.707	0.715	0.585	0.588	0.592
	40	0.705	0.711	0.715	0.585	0.590	0.595
	60	0.715	0.718	0.725	0.585	0.588	0.593
SLACK	20	0.740	0.752	0.770	0.667	0.673	0.677
	40	0.740	0.760	0.780	0.670	0.675	0.682
	60	0.750	0.768	0.785	0.670	0.676	0.678
RND	20	0.735	0.757	0.790	0.527	0.544	0.565
	40	0.720	0.756	0.785	0.517	0.532	0.555
	60	0.725	0.750	0.775	0.508	0.527	0.548

Table 2. Performance of the simulated annealing approach

		Data set 1 (20 jobs)			Data set 2 (60 jobs)		
Rule	Nb size	Min.	Avg.	Max.	Min.	Avg.	Max.
ECR	20	0.765	0.791	0.820	0.805	0.806	0.807
	40	0.765	0.784	0.810	0.805	0.805	0.805
	60	0.765	0.793	0.820	0.805	0.806	0.807
MDD	20	0.740	0.764	0.795	0.655	0.656	0.660
	40	0.745	0.771	0.805	0.655	0.657	0.663
	60	0.745	0.764	0.790	0.655	0.655	0.657
SPT	20	0.720	0.768	0.810	0.575	0.575	0.575
	40	0.740	0.774	0.815	0.575	0.576	0.580
	60	0.745	0.773	0.805	0.575	0.576	0.578
SLACK	20	0.725	0.760	0.810	0.653	0.655	0.658
	40	0.720	0.760	0.800	0.650	0.657	0.667
	60	0.740	0.776	0.810	0.657	0.659	0.667
RND	20	0.735	0.768	0.805	0.455	0.484	0.513
	40	0.725	0.767	0.815	0.462	0.492	0.520
	60	0.740	0.768	0.805	0.463	0.491	0.512

Table 3. Performance of the proposed approach

		Data set 1 (20 jobs)			Data set 2 (60 jobs)		
Rule	Nb size	Min.	Avg.	Max.	Min.	Avg.	Max.
ECR	10	0.875	0.895	0.915	0.843	0.859	0.877
	20	0.875	0.889	0.910	0.840	0.858	0.873
	30	0.880	0.891	0.910	0.847	0.863	0.880
MDD	10	0.835	0.859	0.880	0.702	0.727	0.753
	20	0.830	0.857	0.885	0.700	0.724	0.753
	30	0.830	0.859	0.890	0.697	0.719	0.745
SPT	10	0.815	0.835	0.870	0.632	0.647	0.665
	20	0.815	0.837	0.865	0.637	0.652	0.673
	30	0.815	0.831	0.865	0.627	0.646	0.665
SLACK	10	0.835	0.865	0.895	0.695	0.724	0.750
	20	0.840	0.865	0.890	0.693	0.719	0.750
	30	0.835	0.862	0.885	0.683	0.712	0.738

Table 4. Performance comparison

		Data set 1 (20 jobs)			Data set 2 (60 jobs)		
Search approach	Nb size	Min.	Avg.	Max.	Min.	Avg.	Max.
–	ECR		0.760			0.805	
	MDD		0.645			0.655	
	SPT		0.655			0.575	
	SLACK		0.635			0.650	
iterative improvement	ECR	0.810	0.814	0.820	0.815	0.820	0.825
	MDD	0.750	0.766	0.780	0.675	0.683	0.692
	SPT	0.715	0.718	0.725	0.585	0.590	0.595
	SLACK	0.750	0.768	0.785	0.670	0.675	0.682
simulated annealing	ECR	0.765	0.793	0.820	0.805	0.806	0.807
	MDD	0.745	0.771	0.805	0.655	0.657	0.663
	SPT	0.740	0.774	0.815	0.575	0.576	0.580
	SLACK	0.740	0.776	0.810	0.657	0.659	0.667
memetic algorithm	ECR	0.875	0.895	0.915	0.847	0.863	0.880
	MDD	0.830	0.859	0.890	0.702	0.727	0.753
	SPT	0.815	0.835	0.870	0.637	0.652	0.673
	SLACK	0.835	0.865	0.895	0.695	0.724	0.750

Figures

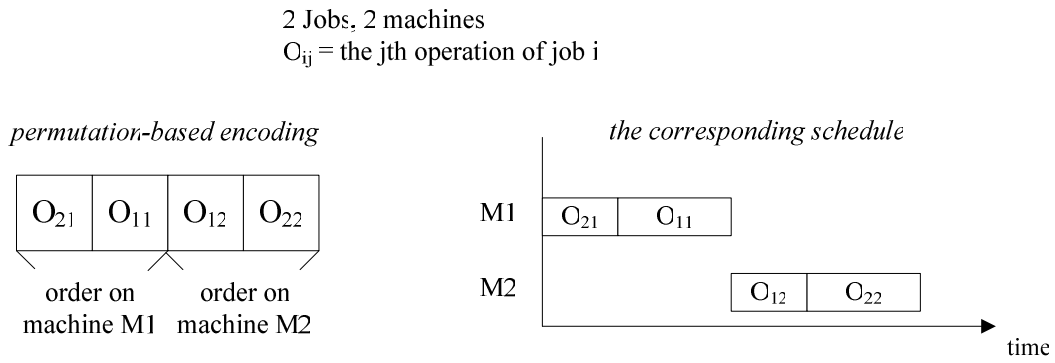

Figure 1. An example of the conventional permutation-based representation

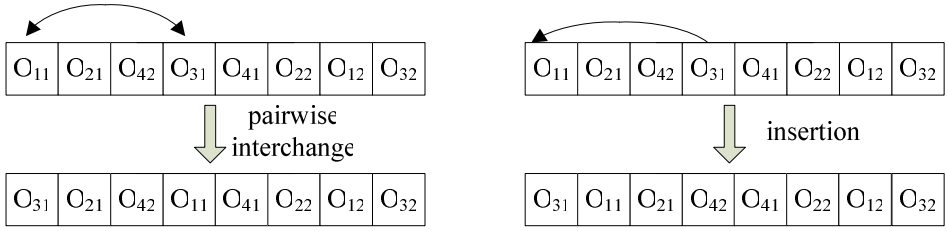

Figure 2. The pairwise interchange and insertion neighborhood function

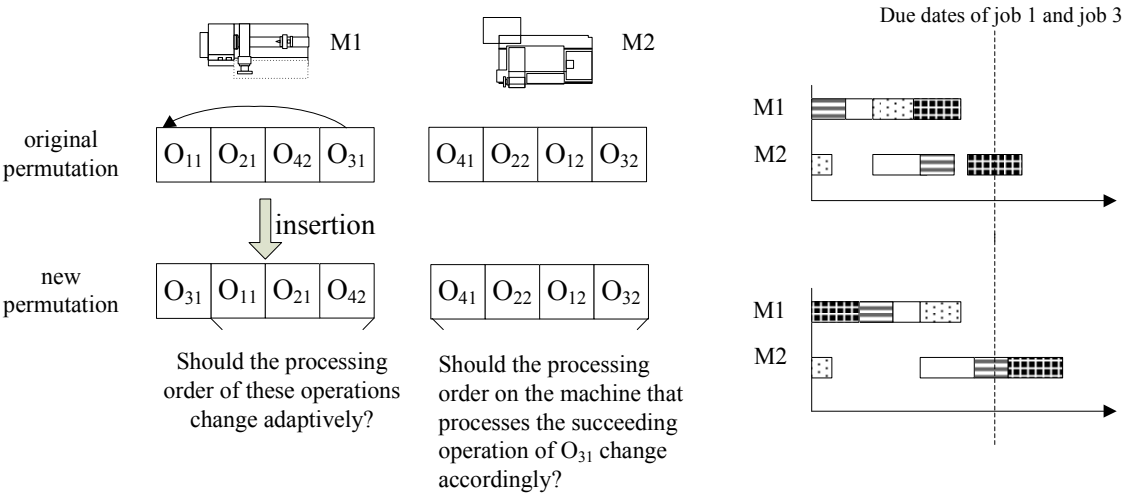

Figure 3. The motivation of the rule-centric concept

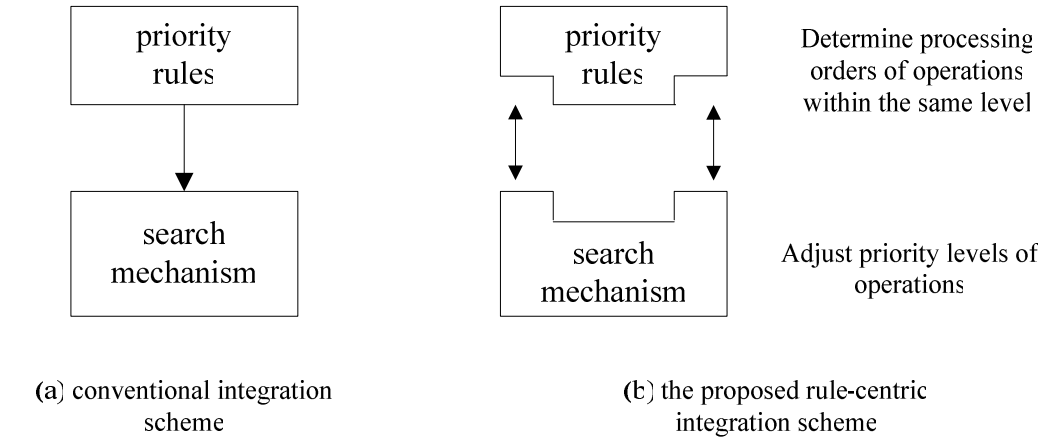

Figure 4. The conventional and the rule-centric integration scheme

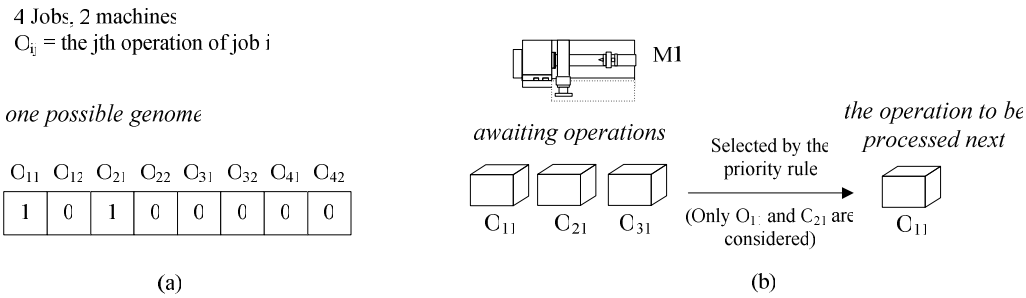

Figure 5. (a) An example of the rule-centric genome representation
(b) Generating a schedule with the representation in (a)

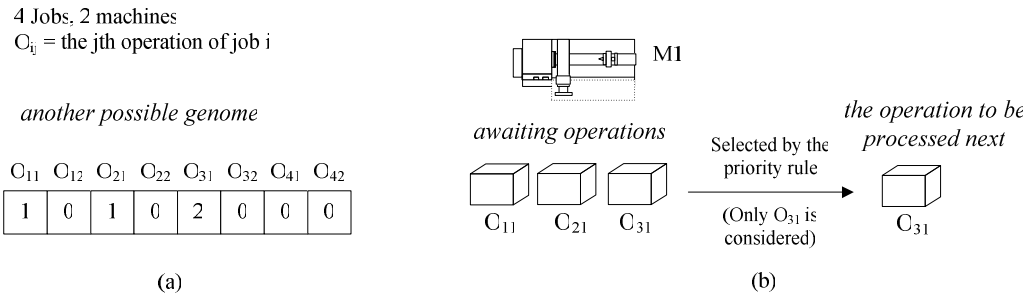

Figure 6. (a) Another example of the rule-centric genome representation
(b) Generating a schedule with the representation in (a)

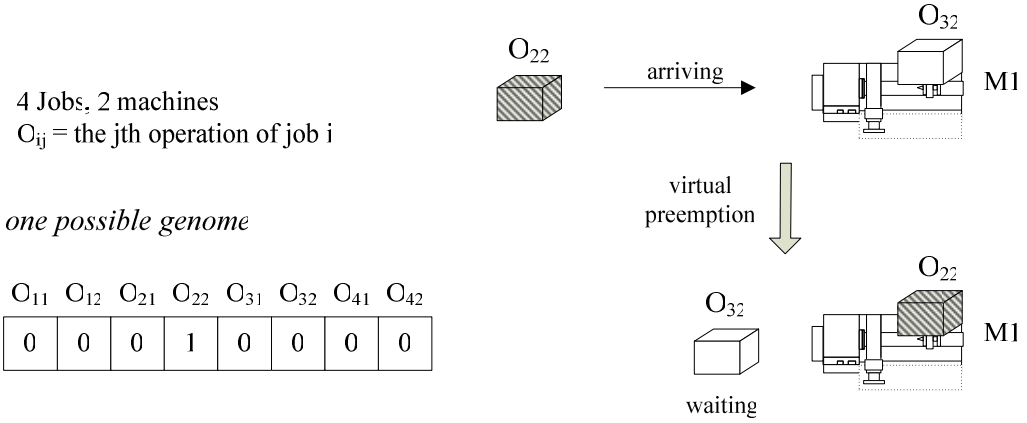

Figure 7. An example of the virtual preemption based on the priority level

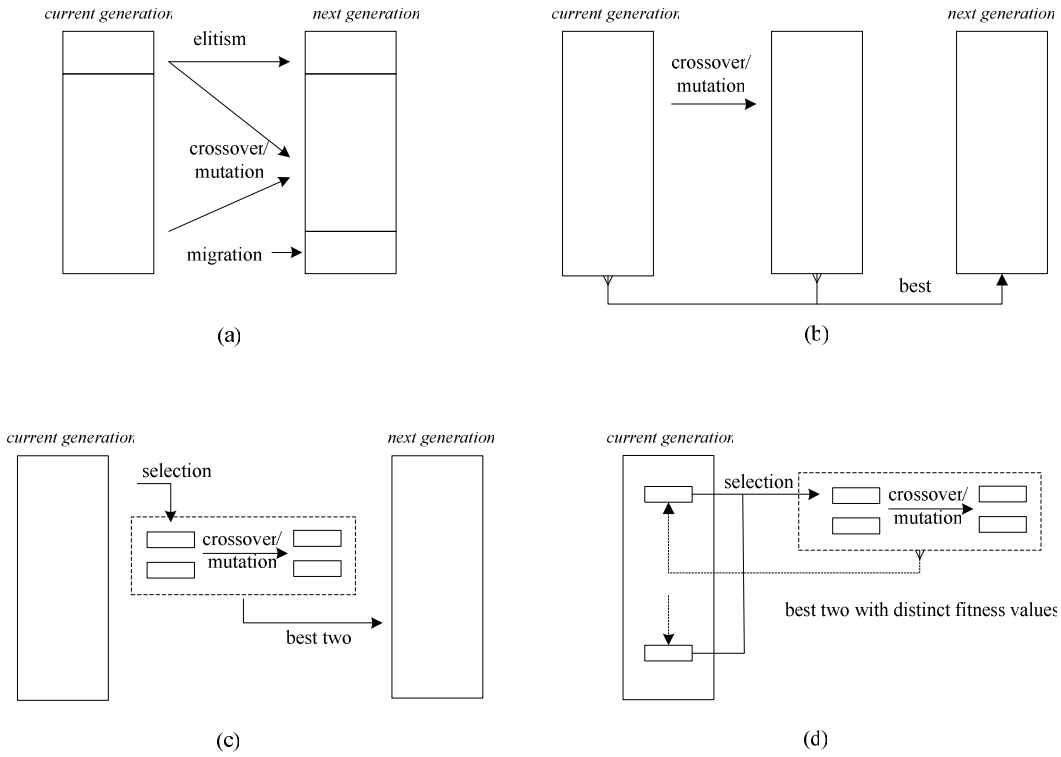

Figure 8. (a) elitism + mating + immigration scheme (b) n/2n scheme (c) 2/4 scheme (d) the proposed scheme

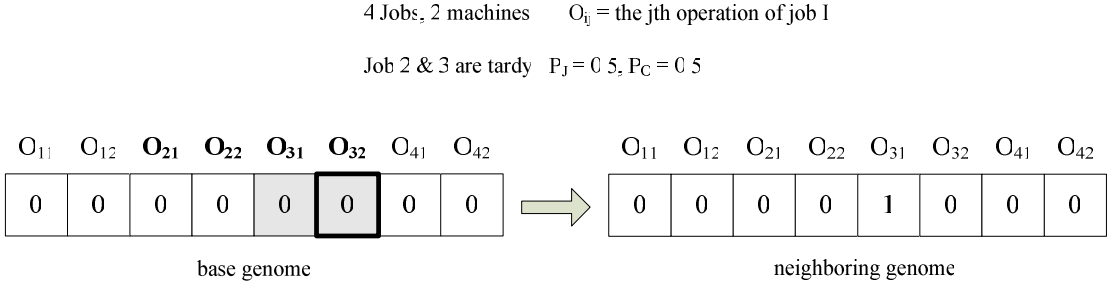

Figure 9. An example of the probabilistic level up (PLU) neighborhood