Abstract

Flexibility, speed, and efficiency are major challenges for operations managers in today’s knowledge-intensive organizations. Such requirements are converted into three production scheduling criteria: (a) minimize the impact of setup times in flexible production lines when moving from one product to another, (b) minimize number of tardy jobs, and (c) minimize overall production time, or makespan, for a given set of products or services. There is a wide range of solution methodologies for such NP-hard scheduling problems. While mathematical programming models provide optimal solutions, they become too complex to model for large scheduling problems. Simultaneously, heuristic approaches are simpler and very often independent of the problem size, but provide “good” rather than optimal solutions. This paper proposes and compares two alternative solutions: 0-1 mixed integer linear programming and genetic programming. It also provides guidelines that can be used by practitioners in the process of selecting the appropriate scheduling methodology.

Keywords: Dual criteria scheduling; Sequence dependent setup times; 0-1 mathematical programming; Genetic algorithm

1. Introduction

Improving scheduling systems for greater customer satisfaction and operations efficiency requires an optimization criterion that incorporates both meeting due dates and minimizing the makespan. Simultaneously, the sequence-dependent setup time environment with dual criteria is a very common scheduling problem in both manufacturing and service organizations [1]. This paper deals with a single-server-scheduling problem with dual criteria: minimizing the number of tardy jobs and makespan, while considering the impact of setup times between jobs. Following the three-field notation provided by Pinedo [2], such a problem is denoted as 1|sjk|\( \sum U_j, C_{\text{max}} \). This dual criteria problem is an NP-hard problem since its simpler case of the single criterion problem, 1|sjk|\( C_{\text{max}} \) is an NP-hard problem [2]. Consequently, this scheduling situation provides an appropriate case study for the comparison of mathematical programming and heuristic programming, two viable alternative solution methodologies.

In general, mathematical programming models ensure the best possible solution for a given scheduling problem. However, due to the complex nature of the problem, such models cannot always be implemented to solve large-scale scheduling problems. On the other hand, heuristic approaches can be used to address combinatorial complexity of NP-hard scheduling problems. The purpose of this paper is to (1) propose two alternative solution methodologies for 1|sjk|\( \sum U_j, C_{\text{max}} \) scheduling problem and (2) compare these two distinct approaches in terms of the appropriateness of each approach under different conditions.

The paper is structured as follows: First, a brief literature review of solution methodologies for bi-criteria, sequence-dependent setup times scheduling is provided.
Then, a 0-1 mixed integer programming model is formulated to solve such a scheduling problem. In Section 4, a genetic algorithm (GA) is proposed to solve the same scheduling problem. In Section 5, we compare two methods and provide guidelines that can be used by practitioners in the process of solving dual criteria scheduling problems with sequence dependent setup times.

2. Literature review

Most of the studies dealing with scheduling have been confined to optimization of a single criterion. Many researchers have considered a variety of scheduling problems with a single performance measure such as the average flow time, maximum completion time, or tardiness [3]. Baker and Scudder [4] classified scheduling problems according to the number of due dates, tightness of due dates and types of penalties involved. The distinct due dates problem without inserted idle time was considered by Abdul-Razaq and Potts [5] and Ow and Morton [6]. Abdul-Razaq and Potts [5] proposed a lower bound scheme based on the branch and bound algorithm. Ow and Morton [6] considered a beam search method in which a limited number of solution paths are investigated in parallel. The intent of the search technique is to search quickly with no backtracking. Thus, the optimal solution is not guaranteed by the method. Fry et al. [7] and Yano and Kim [8] presented algorithms for optimal timing of a given sequence.

Scheduling decisions frequently involve consideration of more than one criterion [9–13]. The bi-criteria scheduling problems are generally divided into two classes. In the first class, the problem involves minimizing one criterion subject to the constraint that the other criterion has to be optimized. The pioneering work in this class can be attributed to Smith [14] who considered the single-machine problem with the maximum tardiness as the primary criterion and mean flow time as the secondary criterion. Extensions of Smith’s work with the consideration of different primary and secondary criterion have been studied by Heck and Roberts [15], Emmons [16], Miyazaki [17], Bianco and Ricciardelli [18], Shanthikumar [19], Chand and Schneeberger [20], and Chen and Bullfin [21]. The problem considered in this paper belongs to this class. We solved it lexicographically by treating the minimization of the number of tardy jobs as the primary objective, and minimization of the makespan as the secondary objective.

In the second class of problems, both criteria are considered equally important and the problem involves finding efficient (non-dominated) schedules. Van Wassenhove and Gelders [22] extended the problem solved by Smith [14], and developed an algorithm which provides an efficient solution for the criteria of mean flow time and maximum tardiness. Several extensions of the single-machine scheduling problem with the consideration of different criteria have been reported [21,23–29].

Since the 1960s, there has been an increasing interest in heuristic techniques, such as simulated annealing, tabu search, and GAs in finding optimal or good solutions to large problems. The term used to refer to such techniques is “evolutionary computation.” The best known algorithms in this class include GAs [30], evolution strategies [31], evolutionary programming [32], and genetic programming [33]. GA has demonstrated their potential for solving difficult intractable optimization problems. GA is proven to be efficient and adaptive even for complex constrained optimization problems. Thus GAs may well be suitable to handle the complexity of multi-criteria scheduling problems [13,34].

Other works in this area include Lee and Choi [35] who considered a job-scheduling problem with distinct due dates in a single machine and Liu and Tang [36] who proposed a modified GA for the single-machine-scheduling problem with ready times. Portman et al. [37] discussed optimal methods for solving k-stage hybrid flowshop scheduling problems and Figielska [38] considered the problem of scheduling preemptive jobs on unrelated parallel machines. The general conceptual design of the GA that we propose in this paper is based on the guidelines provided by Houpt and Houpt [39]. Specific details, such as, incorporating setups, designing crossover operator, and retaining the best solution using “elitism” are based on the work of Rubin and Ragatz [40], Poon and Carter [41], Murata et al. [11], and Neppalli et al. [42].

As noted above, there is a wide range of solution methodologies for solving NP-hard scheduling problems. Recently, there has been an increased interest in comparing the performance of such methodologies under particular scheduling constraints [43–48]. For large and complex scheduling problems, GAs are found to provide optimal or near optimal solutions in a reasonable amount of time [49]. Simultaneously, it is observed that GAs perform less efficiently than integer programming for small to medium sized problems [50]. Continuing this stream of research, we compare our proposed optimal and heuristic solutions with respect to ease of use and quality of the solution.

3. Mathematical formulation for \(1|s|U_j, C_{\text{max}}\)

In this section, we first provide a 0-1 mixed integer linear programming formulation of the dual criteria scheduling with sequence dependent setup times. The objective function includes tardiness and makespan. There is a large number of software designed to solve linear programming models. Considering the availability of the software to scheduling practitioners, we selected Microsoft Excel’s Solver Add-in as a tool to solve and analyze the problem.
3.1. Notations

Indices: $i$ or $j = 1,...,n$ job index used as a unique identifier for each job; $k = 1,...,n$ job index used to identify the position of a job in a given sequence;

Parameters

$p_j$ Processing time for job $j$;
$d_j$ Due time for job $j$;
$S_{0j}$ Setup time of job $j$ in the first sequence position (initial setup time);
$S_{ij}$ Incremental setup time of switching from job $i$ to job $j$;
$M$ A very large number;

Variables

$x_{jk}$ 1 if job $j$ is assigned to the $k$th position in the sequence; 0, otherwise;
$x_{0jk}$ 1 if job $j$ is assigned in the $k$th position and is preceded by job $i$; 0, otherwise;
$U(k)$ 1 if job in the $k$th position is tardy; 0, otherwise;
$C(k)$ Completion time for the job in the $k$th position in the sequence;
$S(k)$ Setup time for the job in the $k$th position in the sequence;
$P(k)$ Processing time for the job in the $k$th position in the sequence; and
$d(k)$ Due time for the job in the $k$th position in the sequence.

3.2. Formulation

Using the above notations, a 0-1 mixed integer LP formulation is presented:

Minimize $Z_1 = \sum_{k=1}^{n} U(k)$  \hspace{1cm} (1)

Minimize $Z_2 = C(n)$  \hspace{1cm} (2)

subject to

\begin{align*}
\sum_{j=1}^{n} x_{jk} &= 1, \quad k = 1,...,n  \\
\sum_{j=1}^{k} x_{jk} &= 1, \quad j = 1,...,n  \\
\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ijk} &= 1, \quad k = 1,...,n  \\
x_{jk} + x_{jk-1} - 1 &\leq x_{jk}, \quad i = 1,...,n, \\
&\quad j = 1,...,n, \quad \text{and} \quad k = 2,...,n  \\
S(1) &= \sum_{j=1}^{n} S_{0j} x_{j1}, \quad \text{subject to} \quad k = 2,...,n  \\
S(k) &= \sum_{j=1}^{n} \sum_{i=1}^{n} S_{ij} x_{jk}, \quad k = 2,...,n  \\
P(k) &= \sum_{j=1}^{n} x_{jk} p_j, \quad k = 1,...,n  \\
C(k) &= C(k-1) + S(k) + P(k), \quad k = 1,...,n  \\
d(k) &= \sum_{j=1}^{n} x_{jk} d_j, \quad k = 1,...,n  \\
−C(k) + d(k) &\leq M(1 - U(k)), \quad k = 1,...,n  \\
C(k) - d(k) &\leq MU(k), \quad k = 1,...,n  \\
U(k), x_{jk}, x_{ijk} &= \{0, 1\}, \quad i = 1,...,n, \\
&\quad j = 1,...,n, \quad \text{and} \quad k = 1,...,n  \\
C(k), S(k), P(k), d(k) &\geq 0, \quad k = 1,...,n  \hspace{1cm} (15)
\end{align*}

Eqs. (1) and (2) are the objective functions. Eq. (1) minimizes the number of tardy jobs and Eq. (2) minimizes the makespan. Eqs. (3) and (4) assure that only one job is assigned to each sequence position and a job is assigned to only one sequence location, respectively. Eqs. (5) and (6) assure that only one job, job $j$, is assigned to follow job $i$. Eq. (7) determines the setup time of the first sequence position while Eq. (8) determines the setup time of the $k$th ($k > 1$) sequence position. Eq. (9) determines the processing time, and Eq. (10) determines the completion time of the $k$th sequence position. Eq. (11) identifies the due date of the job in the $k$th sequence position. Eqs. (12) and (13) identify the tardy positions of the job sequence, where $M$ is a very large positive integer. Finally, Eqs. (14) and (15) represent the integrality and non-negativity constraints.

3.3. Results

From the practitioner perspective, a major concern for the mathematical formulation of this class of scheduling problems would be the number of constraints and decision variables. The model presented above has $n^3 + n^2 + 5n$ variables and $n^3 + 9n$ constraints, where $n$ is the number of jobs which must be sequenced. Fig. 1 graphically represents the change in the number of decision variables and the number of problem constraints when the number of jobs to be scheduled increases.

As shown above, when the number of jobs increases, the number of constraints and decision variables will increase significantly. For example, a five-job scheduling problem requires 175 decision variables and 170 constraints. A 10-job scheduling problem requires 1150 decision variables and 1090 constraints. When more than 20 jobs are considered, the number of decision variables and constraints can become well above ten thousands. Such large models make the formulation and implementation of the proposed mathematical model time consuming and impractical.
However, since the solution derived by such models is optimal, operations schedulers should consider the use of such models when the number of jobs is relatively small. Scheduling complexity in such cases can also be avoided by preparing a user-friendly interface for the purpose of data entering and solution interpretations.

Table 1
Main parameters of GA

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Notation</th>
<th>Description</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Jobs</td>
<td>JOBS</td>
<td>Number of jobs to be sequenced</td>
<td>Independent</td>
</tr>
<tr>
<td>Population Size</td>
<td>POPSIZE</td>
<td>Number of sequences in each generation</td>
<td>Independent</td>
</tr>
<tr>
<td>Weight in the objective function</td>
<td>$W_1$</td>
<td>A value between 0 and 1 used to ponder the value of makespan into the objective function. Since $W_2$ (the weight for tardiness) is a function of $W_1$, the weight for tardiness is omitted from the analysis</td>
<td>Independent</td>
</tr>
<tr>
<td>$W_2 = 1 - W_1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mutation rate</td>
<td>MUTRATE</td>
<td>A value between 0 and 1 which represent the portion of new members in each generation generated using the mutation function</td>
<td>Independent</td>
</tr>
<tr>
<td>Crossover rate</td>
<td>CROSRATE</td>
<td>A value between 0 and 1 which represent the portion of new members in each generation generated using the crossover function</td>
<td>Independent</td>
</tr>
<tr>
<td>Due time ratio</td>
<td>DUERATIO</td>
<td>A random integer ($\geq 1$) which represents an overall ratio between due time and processing time for each job. The larger the value the more likely the due time is significantly greater than processing time</td>
<td>Independent</td>
</tr>
<tr>
<td>Setup time ratio</td>
<td>SETRATIO</td>
<td>A random integer ($\geq 1$) which represents an overall ratio between setup times and processing time for each job. The larger the values the more likely the setup times are significantly greater than processing time</td>
<td>Independent</td>
</tr>
<tr>
<td>Population Vs. jobs</td>
<td>POPJOB</td>
<td>Ratio between population size and number of jobs</td>
<td>Independent</td>
</tr>
<tr>
<td>Mutation Vs. crossover</td>
<td>MUTCROS</td>
<td>Ratio between mutation and crossover rate</td>
<td>Independent</td>
</tr>
<tr>
<td>Number of generations</td>
<td>GEN</td>
<td>Number of generations needed to achieve an optimal solution. The maximum value of this parameter was arbitrary chosen as 1500, i.e. the algorithm was allowed to search for an optimal solution for no more than 1500 generations</td>
<td>Dependent</td>
</tr>
</tbody>
</table>

The above problem was solved for 24 different sets of simulated data using Microsoft Excel’s Solver Add-in. We kept the number of jobs relatively small: from 5 to 8 and data about processing times were normally distributed with an arbitrarily chosen mean of 100 units. As shown in Table 1, due and setup times were randomly generated by using an integer which represents an overall ratio between due (setup) time and processing time for each job. The larger the value of this integer, the more likely the due (setup) time is significantly greater than processing time. As mentioned earlier, we solved this problem by first ignoring the secondary objective. Then the problem is resolved for the secondary objective while the primary objective with its optimal value is treated as a constraint. The same set of data was used to validate the performance of GA as proposed in the following section.

4. GA for $1|\sum U_j, C_{max}$

GAs, when applied to scheduling, view sequences or schedules as individuals, which are members of a population. Each individual is characterized by its fitness value. The fitness of an individual is measured by the associated value of the objective function. The procedure works iteratively and each iteration is referred to as a new generation. The proposed algorithm is shown in Fig. 2.
A generation of individuals consists of surviving individuals of the previous generation and new solutions or offsprings. The population size usually remains constant from one generation to the next. The offspring is generated through reproduction and mutation of individuals (parents) from the previous generation. Individuals at times are referred to as chromosomes. A chromosome may consist of sub-chromosomes, each one containing the information regarding the job sequence on a machine. A mutation in a parent chromosome may be equivalent to a pair-wise interchange in the corresponding sequence. In each generation, the fittest individuals reproduce and the least fit die. As described in the figure, the proposed GA consists of several steps.

Step 1: Create population

- **Define job structure.** Each job consists of several members, such as job name, processing time, due date, and initial setup time.
- **Define sequence structure.** Each sequence consists of two members: (1) an $N$-dimensional array of job structures, and (2) an $N \times N$ setup matrix. Each element of this matrix represents the setup time between job $i$ and $j$, if job $i$ precedes job $j$. Whenever a new permutation is achieved, the setup time matrix must be adjusted accordingly. This is handled by creating a specific adjust_setup_time function. This function uses the list of jobs in a given sequence as a parameter and is called upon every time a crossover or a permutation occurs.
- **Define population structure.** Each generation consists of a vector of sequences of POPSİZE, the population size. For the first generation, applying a mutation operator to the initial sequence creates the initial population. (See details for the mutation operator in Step 3). Subsequent generations are created as described in Step 3 of the algorithm.

Step 2: Evaluate cost. The fitness value or cost of each sequence is used as an optimization objective function for the algorithm. The values of makespan and tardiness are normalized and then factored with respective weights to compute the overall objective function. Using different weights, one may expand the dimensions of the objective function and provide solutions that incorporate multiple objective criteria. The following inductive definitions are used to compute makespan and tardiness for each sequence:

- **Makespan** is the time when the last job in the sequence leaves the system. Makespan can be computed as the completion time of the $n$th job in the sequence.

\[ C(k) = C(k - 1) + S(k) + P(k), \]

where
- $C(k)$ is completion time of the $k$th job in the sequence
- $S(k)$ is the setup time between the $(k - 1)$th and the $k$th job in the sequence
- $P(k)$ is the processing time of the $k$th job in the sequence

- **Tardiness** for each job is computed by comparing the completion time for each job with the due date. For a given job in the sequence, if the due date is greater than the completion time, then the job is considered as tardy.

Step 3: Create new generation. After sorting the population members of the previous generation in an ascending order based on the cost, the process of creating a new generation consist of three main steps:

- **Keep best members.** The process of assigning the first few best members from the old generation to the new generation will ensure a gradual improvement of the solution. The algorithm also saves the best sequence as a candidate optimal solution.
- **Crossover operator.** The following crossover operator code is suggested. This operator generates new sequences and ensures the feasibility of the algorithm.
Step a: Select sequences PARENT 1 and PARENT 2 as two sequences from the old generation.
Step b: Generate $k$ as a random number between 0 and $N$, where $N$ is number of jobs in the sequence.
Step c: Select the first $k$ members of PARENT 1 and save them in the new OFFSPRING.
Step d: Complete the rest $(N - k)$ members of the OFFSPRING by following the following rules: If the rest of the members from PARENT 1 appear in the MOTHER sequence, add the appearing member to the OFFSPRING following the same order they appear in the PARENT 2 sequence.
Step e: Adjust setup matrix for the OFFSPRING sequence.

Mutation operator. The crossover operator is focused on creating alternative solutions around the best solutions achieved so far. In order to avoid the risk of remaining in the local optima, a mutation operator is suggested. For sequencing problems, mutation can be achieved by swapping two random jobs in a given sequence. The algorithm for this process consists of the following steps:
Step a: Randomly generate two integers $k$ and $s$ between 0 and $N$, where $N$ is number of jobs in the sequence.
Step b: Swap jobs that are in the $k$th and $s$th position in a given SEQUENCE.
Step c: Adjust the setup matrix for the new SEQUENCE.

The process of creating new generations continues until a given number of generations is achieved or the cost of a given solution achieves an acceptable level. The genetic program was allowed to run for about 100 generations for the 24 four problems previously solved via mathematical programming. For 21 of such scheduling problems GA achieved the optimal solution within relatively short number of generations. In three cases, the optimal solution was not achieved even after the algorithm exceeded 100 generations.

5. Results of GA

In addition to the above 24 randomly generated problems, we also generated 100 problems for each of the following group of problems: small problems with an average number of jobs of 10 units, problems with an average number of jobs of 15 units, problems with an average number of jobs of 20 units, and lastly large problems with an average number of jobs of 25 units. The standard deviation in all cases was set at 5 units. Again, processing time was generated using a normal distribution with a mean of 100 units. Also, due and setup times were generated using the criteria explained in Table 1. As a result, the total number of cases solved by the GA program was $424 \times 100 + 24$ with a good representation of small, medium and large scheduling problems.

Table 1 also lists other dependent variables which were included in our analysis for the purpose of fine tuning the proposed algorithm. The dependent variable is represented by the number of generations (GEN) needed to achieve an optimal solution. The maximum value of this parameter was arbitrary chosen as 1500. First the algorithm was allowed to search for an optimal solution for up to 1000 generations. This solution was monitored during the next 500 generations and if no improvements were recorded, then we considered that the algorithm has achieved an optimal solution. Otherwise, the problem was disregarded, reducing the total number of cases that were used for further statistical analysis to 204. This approach accentuates GEN as a stronger representation of solution quality and is consistent with the study of Tan et al. [46]. The only difference is that the above mentioned research used computer run time instead of the number of generations (iterations).

Pearson correlation coefficients were calculated to see if there is a linear association between independent variables and number of generations. Table 2 shows the results of this analysis. There are two important findings to mention by analyzing this table. First, there is no significant correlation between the number of jobs to be sequenced and the number of generations required to derive a solution. Thus, contrary to the mathematical formulation, we can conclude that the number of jobs does not affect the performance of the algorithm. This is a significant advantage of the proposed GA algorithm as compared to the previous mathematical programming model used to solve this sequencing problem.

Second, there is a significant indirect correlation between setup time and the number of generations. The algorithm tends to perform better when setup times between consecutive jobs increase for the same processing times. In order to increase the reliability of our analysis, we performed a discriminant analysis. In addition, we added other dependent variables in our analysis, as shown in Table 3.

This table shows significance tests for equality of group means for three parameters of the algorithm. The algorithm shows significantly better performance when the population size selected increases over the same number of jobs ($0.007 < 0.01$). The same can be concluded for the crossover rate, where the observed significance level is 0.05.

6. Conclusions and recommendations

This paper provides two alternative solution methodologies for an NP hard scheduling problem: single machine, sequence dependent-setup time, and dual criteria. The two alternative methodologies are a 0-1 mixed integer-programming and a genetic search algorithm. Due to increasing computer presence and its processing power in today’s workplace, operation schedulers’ concern is not the time that a computer takes to generate a solution. Instead,
Table 2
Pearson correlation coefficients for GA

<table>
<thead>
<tr>
<th>JOBS</th>
<th>POPSIZE</th>
<th>W1</th>
<th>MUTRATE</th>
<th>CROSRATE</th>
<th>DUERATIO</th>
<th>SETRATIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEN</td>
<td>Coefficient</td>
<td>0.063</td>
<td>0.075</td>
<td>0.104</td>
<td>0.030</td>
<td>0.031</td>
</tr>
<tr>
<td>Significance</td>
<td>0.370</td>
<td>0.284</td>
<td>0.137</td>
<td>0.670</td>
<td>0.066</td>
<td>0.694</td>
</tr>
</tbody>
</table>

Table 3
Tests of equality of group means

<table>
<thead>
<tr>
<th></th>
<th>Wilks’ lambda</th>
<th>F</th>
<th>df1</th>
<th>df2</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>POPJOB</td>
<td>0.925</td>
<td>7.688</td>
<td>1</td>
<td>95</td>
<td>0.007</td>
</tr>
<tr>
<td>MUTRATE</td>
<td>0.969</td>
<td>3.043</td>
<td>1</td>
<td>95</td>
<td>0.084</td>
</tr>
<tr>
<td>CROSRATE</td>
<td>0.944</td>
<td>5.644</td>
<td>1</td>
<td>95</td>
<td>0.020</td>
</tr>
</tbody>
</table>

Table 4
Practical recommendations for \(1|s_{ij} \sum U_j C_{max}\) scheduling problem

<table>
<thead>
<tr>
<th>IF goal is finding (a(n)):</th>
<th>AND IF number of jobs ranges:</th>
<th>AND IF ratio between setup and processing times is:</th>
<th>THEN solution methodology suggested is:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal solution</td>
<td>Less than 5</td>
<td>High</td>
<td>0-1 mathematical programming</td>
</tr>
<tr>
<td></td>
<td>From 6 to 10</td>
<td>Low</td>
<td>0-1 mathematical programming</td>
</tr>
<tr>
<td></td>
<td>Above 11</td>
<td>High</td>
<td>Genetic algorithm</td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td></td>
<td>0-1 mathematical programming</td>
</tr>
<tr>
<td>Practical solution</td>
<td>Less than 5</td>
<td>High</td>
<td>0-1 mathematical programming</td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td></td>
<td>0-1 mathematical programming</td>
</tr>
<tr>
<td></td>
<td>From 6 to 10</td>
<td>High</td>
<td>Genetic algorithm</td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td></td>
<td>Genetic algorithm</td>
</tr>
<tr>
<td></td>
<td>Above 11</td>
<td>High</td>
<td>Genetic algorithm</td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td></td>
<td>Genetic algorithm</td>
</tr>
</tbody>
</table>

more practical goals would be to improve user interface and to achieve the best possible solution. Table 4 provides some practical recommendations that can be used to select one of the above two solution methods.

For the \(1|s_{ij} \sum U_j C_{max}\) scheduling problem, our findings suggest that when the major concern of the operations scheduler is the quality of the output, and when the number of jobs to be sequenced is relatively small, then mathematical programming is an appropriate methodology. While mathematical programming generally ensures an optimal solution, genetic algorithms do not always guarantee an optimal solution. If the number of jobs to be sequenced is relatively large or when achieving a “good” solution is acceptable, then genetic algorithms (GAs) can be used. The number of jobs that need to be sequenced does not affect the GA performance. In contrast, mathematical programming model becomes very complex and even unmanageable when the number of jobs is more than ten. In addition, our analysis shows that the proposed GA performs better when the ratio between sequence-dependent setup times and processing times for each job is relatively large.

References


