Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/080562

A Job Shop Scheduling Game with GA-based Evaluation

Kim Jun Woo[*]

Department of Industrial and Management Systems Engineering, Dong-A University, Busan, Korea

Received: 4 Oct. 2013, Revised: 1 Jan. 2014, Accepted: 2 Jan. 2014
Published online: 1 Sep. 2014

Abstract: The job shop scheduling problem is one of the well-known hardest combinatorial optimization problems, and solving the problem can be complex and time-consuming task. Hence, many undergraduates can not fully experience the scheduling procedures. This paper aims to introduce a computer game designed to enable the players to create and solve the job shop scheduling problems, and the players can learn the topics such as scheduling and optimization via a sense-making experiences provided by the game playing. The game program provides a simple graphic user interface similar with traditional board games played by manipulating blocks, and the players can conveniently search the good schedules in a trial-and-error manner. Moreover, the completed schedules are evaluated by taking the schedules generated by the genetic algorithm into account. For reasonable evaluation, the game program adopts two search strategies called forward and backward search to generate schedules with a wide range of makespans. The integrated job shop scheduling game introduced is well-organized to support overall procedures for job shop scheduling, and the genetic algorithm based evaluation can make the players to compete for better schedules. As a result, both purposes, entertainment and education, can be simultaneously served by the game playing.

Keywords: computer game, serious game, job shop scheduling, genetic algorithm, industrial engineering education

1 Introduction

Gaming is one of the most popular activities for entertainment and fun today, and provides the players with highly immersive experiences [1]. This immersion has led many researchers to argue that the use of games can enhance the students learning [2]. Today, it is generally accepted that the game-based learning is very effective learning strategy, and the educational games are widely studied and adopted for learners range from children to undergraduates and adults [3].

Especially, the proliferation of management and industrial engineering courses created a context where games could be adopted for professional knowledge and skills at the university or higher levels, and many educational games called operations management games have been introduced in these domains [4]. For example, the famous Beer game provides a good insight into the concepts and the problems such as bullwhip effect in the supply chain management [5]. Although the concepts and procedures of operations management activities are hard to be experienced by the undergraduates in general, such games provide good alternative sense-making experiences that promote the learners learning [1].

This paper aims to introduce a well-organized operations management game called the integrated job shop scheduling game, which enable the players to learn the concept of production scheduling by game playing. Although the job shop scheduling problem (JSP) is an important research topic in the domain of management and industrial engineering, it is one of the well-known hardest combinatorial optimization problems, and manually constructing the schedules for these problems is complex and time-consuming task [6, 7]. These have been the main obstacles that make it difficult for the students to learn and practice the scheduling procedures.

The previous simple job shop scheduling game [8] demonstrated a conceptual design for the graphic user interface based game where the goal of the players is to construct good schedules by manipulating blocks. However, there has been an important limitation that the simple job shop scheduling game can not evaluate the schedules constructed by the players, that is, the players can not see if they achieve good schedules or not.

The integrated job shop scheduling game introduced in this paper is an enhanced version of the previous simple job shop scheduling game, where the players can create their own JSP and the schedules constructed by

[*] Corresponding author e-mail: kjunwoo@dau.ac.kr

them can be evaluated. To this end, the integrated job shop scheduling game searches a variety of schedules with a wide range of makespans for a given JSP by applying genetic algorithm.

The genetic algorithm, which simulates the genetic process of biological organisms in nature, is one of the meta-heuristic methods appropriate for solving the combinatorial optimization problems such as JSP [9,10]. In addition, the genetic algorithm maintains the solution population including various schedules in searching, and this led the integrated job shop scheduling to adopt the genetic algorithm to evaluate the players schedules.

The remainder of this paper is organized as follows: Section 2 provides a literature review on related works, and the overall structure of the integrated job shop scheduling game is explained in Section 3. The experiment results are represented in Section 4, and finally, the concluding remarks and the future research directions follow in Section 5.

2 Research Backrounds

2.1 Job Shop Scheduling Problem

Scheduling of operations, determining the start and the finish time of each operation, is an important task in the production planning and operations management [11]. The JSP is one of the most well-known scheduling problems, characterized by n jobs to be processed on distinct m machines. Each job is composed of m operations which must be processed in a pre-specified order. Moreover, a single machine can process only one operation at a time, and the operations can not be interrupted. In general, the goal of JSP is to obtain the optimal schedule, composed of the start time and finish time of each operation, which minimize the makespan, the maximum finish time [12,13].

A specific JSP can be defined by the problem size, n and m, and the processing times t_{ij}s and the processing machines m_{ij} s of the operation o_{ij}s, where i and j denote the job number and the operation number, respectively. For example, a 3 by 3 JSP is represented in Fig. 2.

The schedules, the solutions of a JSP, are constructed by determining the start time s_{ij} and the finish time f_{ij} of o_{ij} (i=1,2,$\cdots$,n, j=1,2,$\cdots$,m), and they are represented by using a diagram called Gantt-chart as shown in Fig. 5. That is, a schedule can be constructed by creating a corresponding Gantt-chart.

It is straightforward that an operation is represented as a rectangle in a Gantt-chart. Moreover, the heights of all rectangles are identical and the width of a rectangle is proportional to the processing time of corresponding operation. Therefore, it can be said that a schedule is constructed by placing the 'blocks', rectangles with predetermined sizes, on the empty Gantt-chart appropriately, and this is the basic concept of the previous simple job shop scheduling game [8].

2.2 Genetic Algorithm

Although the players of the simple job shop scheduling game can construct the schedules for a given JSP by manipulating the 'operation blocks' conveniently, there is an important limitation that the evaluation for a constructed schedule is not provided. In this context, the main objective of the integrated job shop scheduling game introduced in this paper is to provide reasonable evaluation for the schedules constructed by the players.

It is well known that the JSP is one of the hardest NP-hard problems, and this led many researchers to apply the meta-heuristic methods such as genetic algorithm, tabu search and simulated annealing for solving JSPs [6, 10,12]. In general, such meta-heuristic methods aim to search the optimal or near-optimal solutions via iterative procedures, where a variety of solutions are found, and the main idea of the evaluation of the integrated job shop scheduling game is that a specific solution can be evaluated by comparing it to these solutions. To this end, the integrated job shop scheduling game adopts the genetic algorithm since it maintains the population of many solutions in search procedure.

However, there are two problems to be resolved to use the genetic algorithm for evaluation. First, the conventional genetic algorithms for the JSP generally require relatively long processing times [14]. Since the optimal schedules belong to active schedules, the genetic algorithms for solving JSP typically concentrate on producing active schedules [15]. The Giffler-Thompson algorithm has been widely used in the previous genetic algorithms for this purpose, although it requires a significant processing time [16,17]. On the contrary, the integrated job shop scheduling game adopts semi active schedule based genetic algorithm called sa-GA. The genetic operators, crossover and mutation, of the sa-GA are similar with those of traditional GT/GA algorithm [12], however, the sa-GA represents a schedule as a permutation of operations, which is simply decoded into a semi-active schedule.

The second problem is that the search of the genetic algorithm is directed to focus on the schedules with shorter makespans, although both good and bad schedules are useful for the purpose of evaluation. Therefore, the integrated job shop scheduling game uses two search strategies called forward search and backward search. The forward search aims to find good schedules with shorter makespans, and this can be achieved by using traditional fitness functions used in previous genetic algorithms for solving JSP. On the contrary, the backward search aims to find bad schedules with longer makespans, and modified fitness function is used in this strategy.

3 Integrated Job Shop Scheduling Game

Overall, the integrated job shop scheduling game consists of 4 phases as shown in Fig. 1. First, the game enables the

users to create user-defined job shop scheduling problems so that they can deal with a variety of problems. After a problem is created, the operation blocks and the game board are prepared and the users can play the game by trying to construct a schedule with shorter makespan. In addition, the users must explore the solution space to obtain evaluation result for their schedule. For a single problem, a single solution exploration is required. If a user completes building a schedule and solution exploration has been done, the game program provides an evaluation results for the schedule so that the user can see how good his or her schedule is. Of course, the users can adjust their schedule after evaluation to find an optimal schedule in a trial-and-error manner.

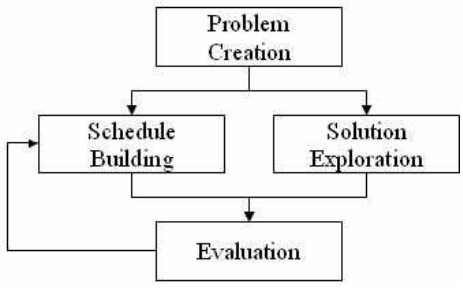

Fig. 1: Overall procedure of the game playing

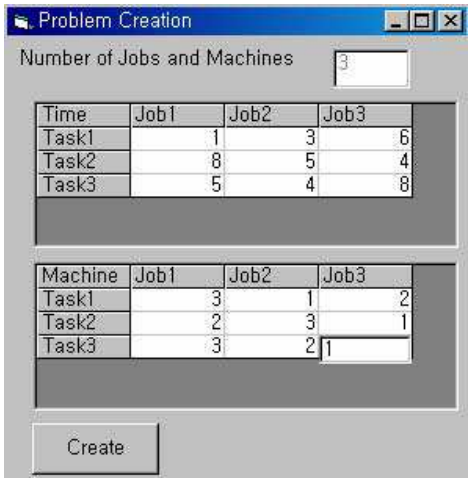

Fig. 2: Problem Creation Phase

3.1 Problem Creation

The users of the integrated job shop scheduling game can create their own problems to tackle in game playing. Since a JSP can characterized by problem size, processing times of the operations, and processing machines of the operations, the users must specify these values to create a problem. Fig. 2 shows the 'problem creation window' for this purpose.

If all required values are specified, the users click the 'create' button in the bottom of the window, and the game program prepares for the schedule building phase.

3.2 Schedule Building

If a problem is created, the main game window is initialized as shown in Fig. 3. Note that the game window in Fig. 3 is initialized by the job shop scheduling problem in Fig. 2. The main game window consists of two major parts, 'game board' in the upper part and 'operation block box' in the lower part.

In the operation block box, there are several rectangles, operation blocks which corresponds to the operations of the current problem. The operation blocks in the same job are placed in the same row with same color. For example, three green blocks in the upper part of the operation block box corresponds to three operations in the Job 1. The heights of all operation blocks are identical, however, the width of operation block o_{ij} is proportional to t_{ij}. In addition, the job number, the operation number, the machine number and the processing time are specified on each operation block.

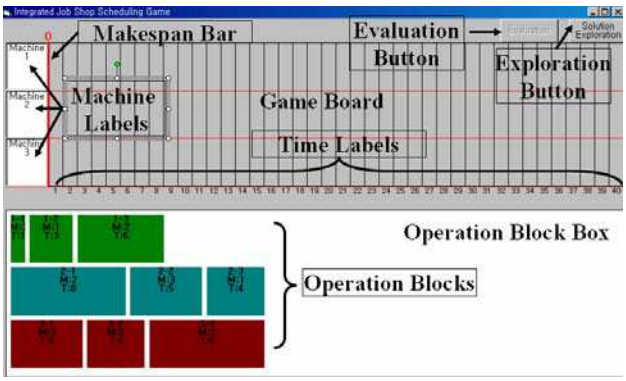

Fig. 3: Initialized Game Window

To manipulate an operation block, player must select the block by clicking it, and then, the font-color of the selected block becomes white. If a selected block is clicked again, it is deselected and the font-color becomes black.

The initial game board corresponds to an empty Gantt-chart, where horizontal axis denotes time and vertical axis denotes the machines. The player can place the selected operation block on the game board by clicking appropriate time label at the bottom of the game board. If a operation block o_{ij} is selected and the player clicks the time label t,

this manipulation makes the block to occupy the (t,m_{ij}), ($t+1$, m_{ij}), $\cdots$, ($t+t_{ij}-1$, m_{ij}) cells in the game board. At the same time, s_{ij} and f_{ij} are determined as follows:

$$s_{ij} = t-1, f_{ij} = t+t_{ij}-1. \quad (1)$$

For example, if a player selects o_{21} block and clicks time label 1, the block is placed on the game board as shown in Fig. 4, and the program sets s_{21} =0 and f_{21}=8.

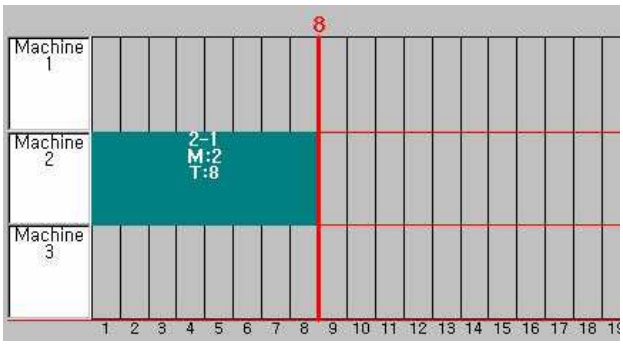

Fig. 4: Placement of Operation Block

Each placement of an operation block can cause a movement of the makespan bar in the game board, which indicates the maximum finish time=$max(f_{ij})$. If all operation blocks are appropriately placed on the game board, the makespan bar indicates the makespan of the completed schedule, and the color of the font-color of the makespan bar becomes blue as shown in Fig. 5, where the makespan=26. Otherwise, the font-color of the makespan bar is red as shown in Fig. 4, and the player can check the length of the current schedule under construction by looking at the makespan bar.

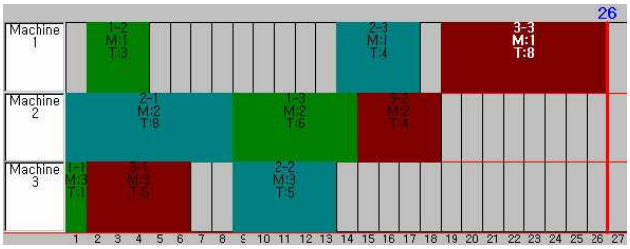

Fig. 5: Completed Schedule

In addition, the positions of the operation blocks placed on the game can be adjusted by selecting a block and clicking other time label, and the players can move them back to the operation block box by selecting a block and clicking the operation block box. In this way, the players can search the optimal solution by manipulating the operation blocks in a trial-and-error manner.

Table 1: Function *isOverLap*

```
boolean isOverLap(i, j)
{
  for a = 1 to n
    for b = 1 to m
      if m_ij=m_ab and o_ij ≠ o_ab and o_ab is on the board
        if (s_ij > s_ab and s_ij < f_ab) or
        (f_ij > s_ab and f_ij < f_ab) or
        (s_ij ≤ s_ab and f_ij ≥ f_ab) return true;
  return false;
}
```

Table 2: Function *isInfeasible*

```
boolean isInfeasible(i, j)
{
  for a = 1 to j-1
    if s_ij < f_ia return true;
  for a = j+1 to m
    if o_ab is on the boad and f_ij > s_ia return true;
  return false;
}
```

Meanwhile, a manipulation of an operation block can be invalid, and in this case, the game program should cancel the current manipulation and restore the previous game state. The invalid manipulation causes violations of the constraints of the JSP, and can produce infeasible schedule, so they must be restricted.

The first constraint of the JSP is that two or more operation blocks can not overlap at all, since each machine can handle one operation at a time. In the integrated job shop scheduling game, the violation of this constraint of the operation block o_{ij} is checked by the Boolean function *isOverLap* in Table 1.

The second constraint is the precedence in a single job. That is, operation block must start after max($f_{i1}, f_{i2}, \cdots, f_{ij-1}$), and must be finished before min($f_{ij+1}, f_{ij+2}, \cdots, f_{im}$). The violation of the precedence constraint of the operation block o_{ij} is checked by the Boolean function *isInfeasible* in Table 2.

Overall, the schedule building phase of the integrated job shop scheduling game forms most part of the game playing, and it provides the users with experience similar with previous simple job shop scheduling game.

3.3 Solution Exploration

The users can perform solution exploration whenever after a problem is created and initialized as shown in Fig. 3, even if a schedule is not completed, by clicking the exploration button. After a single execution of this phase, the exploration button is disabled.

The objective of this phase is to create a list of schedules with a wide range of makespan values, and this list will be used to approximate the percentiles of the schedules constructed by the players. That is, the constructed schedules are evaluated by comparing them to the schedules included in the schedule list produced in solution exploration phase. The schedule list is created by applying a well known meta-heuristic method, genetic algorithm. Moreover, since both good and bad schedules are useful for the purpose of evaluation, two search strategies called forward search and backward search are adopted in creating the schedule list. The solution exploration phase is summarized in Fig. 6.

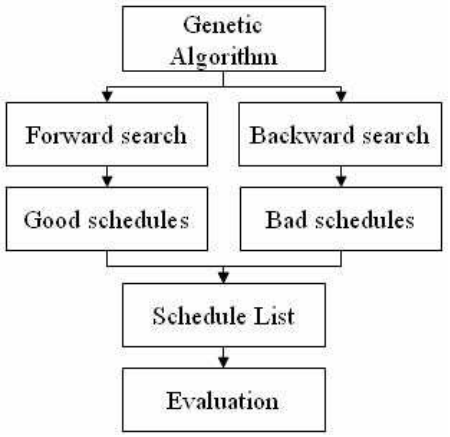

Fig. 6: Solution Exploration Phase

Among the genetic algorithms for solving JSP, the integrated job shop scheduling game adopts the sa-GA. The sa-GA represents a schedule as an operation assignment sequence, which is easily decoded into a semi active schedule. For illustration, lets consider an operation assignment sequence in (2), for the JSP in Fig. 2. What is important is that the decoded schedule is constructed by assigning an operation at a time, and an operation can not precede any previously assigned ones.

$$o_{31}o_{21}o_{22}o_{32}o_{11}o_{12}o_{13}o_{33}o_{23} \tag{2}$$

Note that all operation assignment sequence must not violate the precedence constraint, and in decoding, the start time and the finish time of the k th operation o_{ij} in a operation assignment sequence are determined as follows:

$$s_{ij} = max(M, J) \tag{3}$$

$$f_{ij} = s_{ij} + t_{ij}, \tag{4}$$

where M is the maximum finish time of the previously assigned operations with processing machine= m_{ij} and J is the maximum finish time of the previously assigned operations in job i. hence, the operation assignment sequence in (2) is decoded into the schedule in Table 3.

Table 3: An Example of Decoded Schedule

operation	start time (s_{ij})	finish time (f_{ij})
o_{31}	0	5
o_{21}	0	8
o_{22}	8	13
o_{32}	8	12
o_{11}	13	14
o_{12}	14	17
o_{13}	17	23
o_{33}	17	25
o_{23}	25	29

Table 4: The Parent Operation Assignment Sequences

ID	Operation assignment sequence
P_1	$o_{31}o_{21}o_{22}o_{32}o_{11}o_{12}o_{13}o_{33}o_{23}$
P_2	$o_{31}o_{32}o_{33}o_{21}o_{22}o_{23}o_{11}o_{12}o_{13}$

Table 5: The Child Operation Assignment Sequences

ID	Operation assignment sequence
C_1	$o_{31}o_{32}o_{21}o_{33}o_{22}o_{23}o_{11}o_{12}o_{13}$
C_2	$o_{31}o_{21}o_{32}o_{22}o_{33}o_{11}o_{23}o_{12}o_{13}$

Indeed, using the operation assignment sequence in (2) and simple decoding method can produce semi active schedules, however, as the population evolves, the active schedules are preferred and the proportion of the semi active schedules remains low. Since the sa-GA does not aim to construct active schedules, it can explore the solution space in relatively short time. Moreover, it can search a wide range of makespans effectively in that both active and semi active schedules will be found.

The genetic operators, crossover and mutation, of sa-GA are similar with those of the traditional GT/GA algorithm. The crossover operator creates a child operation assignment sequence from given two parents by assigning an operation at a time. The operation to assign is selected by the parents among the assignable operations. Note that an operation is assignable if and only if there is no preceding operation not assigned. For example, lets consider two parents in Table 4.

The uniform crossover of the sa-GA can be performed as follows: the child C_1 is constructed by assigning an operation selected by P_1, an operation selected by P_2, an operation selected by P_1, $\cdots$, at a time. Similarly, the child C_2 is constructed by assigning an operation selected by P_2, an operation selected by P_1,an operation selected by P_2, $\cdots$, at a time. As a result, we can obtain two child operation assignment sequences in Table 5 from the parents in Table 4.

The mutation operator is also done in assigning an operation to an operation assignment sequence by selecting an operation not selected by any parent sequences. For example, both P_1 and P_2 select o_{31} for the first assignment from the assignable operations, o_{11}, o_{21} and o_{31}. If the mutation is applied to the first assignment, one of the operations not selected, o_{11} and o_{21}, is randomly chosen.

To create the schedule list, the game program adopts two search strategies, forward search and backward search. Both search strategies start with same initial population but they use different fitness functions. In the forward search, a schedule with a shorter makespan should be preferred so that optimal or nearly-optimal solutions will be included in the schedule list. This can be achieved by using ordinary fitness function for the JSP as in (5).

$$fitness\ value = \frac{1}{makespan} \tag{5}$$

On the contrary, the backward search aims to find bad schedules, so a schedule with a longer makespan should be preferred. The found bad schedules are also included in the schedule list, and this can be achieved by using fitness function as in (6). As a result, we can obtain various schedules and their makespans after the solution exploration phase.

$$fitness\ value = makespan \tag{6}$$

3.4 Evaluation

If the solution exploration is done and the player completes a schedule, he or she can evaluate the schedule by clicking the evaluation button. Then, the game program approximates the percentile of the makespan of the schedule constructed by the player as in (7), where $n(worseschedule)$ is the number of schedules in the schedule list with makespans larger than the makespan achieved by the player.

$$percentile = \left(1 - \frac{n(worse\ schedules)}{n(schedule\ list)}\right) \times 100 \tag{7}$$

If the makespan achieved by the player equals to the best makespan in the schedule list, the constructed schedule may be an optimal schedule. Since this evaluation method is based on the schedule list produced in the solution exploration phase, the schedule list must include a variety of schedules with a wide range of makespan values. Of course, the schedule list should contain all possible schedules to make accurate percentile of the players result, however, this is time-consuming and sometimes impossible in that the number of schedules explosively increases as the size of the JSP grows. On the contrary, the integrated job shop scheduling game adopts the genetic algorithm to create the schedule list, and good and bad solutions can be effectively found in a reasonable time.

Table 6: Makespans of the Generated Sequences

makespan	number of sequences
24	9
25	28
26	20
27	6
28	8
29	7
30	12
31	1
33	8
34	14
35	5
38	1
39	1
43	2
44	1

Table 7: The Best 9 Operation Assignment Sequences

ID	Operation Assignment Sequence
1	$o_{31}o_{21}o_{32}o_{11}o_{22}o_{12}o_{33}o_{13}o_{23}$
2	$o_{21}o_{31}o_{11}o_{32}o_{22}o_{12}o_{13}o_{33}o_{23}$
3	$o_{21}o_{31}o_{11}o_{32}o_{12}o_{22}o_{13}o_{33}o_{23}$
4	$o_{21}o_{31}o_{11}o_{32}o_{12}o_{33}o_{22}o_{23}o_{13}$
5	$o_{21}o_{31}o_{11}o_{32}o_{12}o_{33}o_{22}o_{13}o_{23}$
6	$o_{31}o_{21}o_{11}o_{32}o_{12}o_{22}o_{13}o_{33}o_{23}$
7	$o_{21}o_{31}o_{11}o_{12}o_{22}o_{32}o_{33}o_{23}o_{13}$
8	$o_{31}o_{21}o_{11}o_{32}o_{12}o_{33}o_{22}o_{23}o_{13}$
9	$o_{31}o_{21}o_{32}o_{11}o_{12}o_{22}o_{33}o_{13}o_{23}$

4 Experiment Result

For illustration, the JSP in Fig. 2 is created and the schedule list is obtained by the solution exploration phase. To execute the genetic algorithm, we use the population size=20, crossover rate=0.5 and mutation rate=0.01. After the 20 iterations of the forward and the backward sa-GA, 123 distinct operation assignment sequences have been found, and the distribution of their makespans is summarized in Table 6.

Note that the schedules in the schedule list are in the form of the operation assignment sequence, and two or more assignment sequences can be decoded into the identical schedule. For example, 9 best operation assignment sequences are listed in Table 7, and these are encoded an identical schedule shown in Fig. 7.

The evaluation button provides an approximate percentile of the schedule constructed by the player based on the schedule list summarized in Table 6. For example, lets assume that a player constructed a schedule shown in

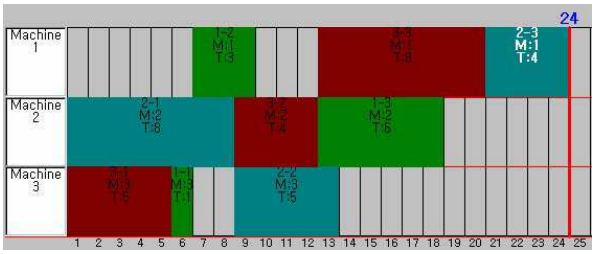

Fig. 7: The Best Schedule

Fig. 5. Then, the approximate percentile of the constructed schedule is 46, and the player can see that the schedule is among the top 46% of the schedules. Since this is not optimal, the player can aim to find better schedule by continuous game playing.

If a player constructs the optimal schedule shown in Fig. 7, the approximate percentile is 7, and this may be an optimal schedule since no schedule included in the schedule list is better than the constructed one. In this case, the game program informs the player that the current schedule may be an optimal, and the current JSP is cleared. In this way, the integrated job shop scheduling game enables the player to create a JSP and find good schedules by playing the game, and it can evaluate the performance of the player. This game can provide fun experiences in that the players compete for the better schedule by manipulating the operation blocks. Moreover, the players can learn about the topic of job shop scheduling via sense-making experiences and the integrated job shop scheduling game will be helpful for the undergraduates in the department of industrial engineering or management.

On the contrary, we can see an important limitation of the game. The evaluation phase is based on the schedule list produced by solution exploration, and it is straightforward that the schedule list should contain schedules representing the various possible schedules. However, in Table 6, there are too many schedules with relatively short makespan, from 24 to 26. Many of them may be found by the forward search, and this aspect can make the game program to underestimate the schedules constructed by the players. Hence, a novel search strategy may be required to create more appropriate schedule list.

5 Conclusions

The JSP is one of the well-known hardest combinatorial optimization problems, and it is an important topic in the industrial engineering education. However, solving JSP can be complex and time-consuming task even if the size of the problem is relatively small, and many undergraduates do not experience the scheduling procedure enough.

Instead, this paper introduced an integrated job shop scheduling game which enables the players to create problems and construct schedules, which can be evaluated by the game program. Since the game provides fun experiences which can attract the students interests. Moreover, the sense-making experiences provided by the simple graphic user interface will be very helpful for learning the job shop scheduling.

However, further topics still need to be investigated. The schedules included in the schedule list are found by a stochastic search method, genetic algorithm, and the schedule list must represent the characteristics of the population of all possible schedules. To this end, the integrated job shop scheduling game adopts two search strategies, forward search and backward search, but the produced schedule list seems to contain too many good schedules. Therefore, appropriate search strategies for a variety of JSP should be studied in future research.

Acknowledgement

This work was supported by the Dong-A University research fund.

References

[1] Kim J. W. and Ha S. H., The Journal of Future Game Technology, **1**, 87-96 (2011).
[2] Kim H. T., Ying K. T. and Pui L. C., Computers and Education, **55**, 109-117 (2010).
[3] N. Vos, H. V. D. Meijden and E. Denessen, Computers and Education, **56**, 127-137 (2011).
[4] M. A. Lewis and H. R. Maylor, International Journal of Production Economics, **105**, 134-149 (2007).
[5] J. S. Goodwin and S. G. Franklin, Journal of Management Development, **13**, 7-15 (1994).
[6] R. Cheng, M. Gen and Y. Tsujimura, Computers and Industrial Engineering, **30**, 983-997 (1996).
[7] F. Pezzela, G. Morganti and G. Ciaschetti, Computers and Operations Research, **35**, 3202-3212 (2008).
[8] Kim J. W. and Sok Y. Y., The Journal of Future Game Technology, **2**, 165-170 (2012).
[9] J. H. Holland, Scientific American, **267**, 66-72 (1992).
[10] J. A. Ruiz-Vanoye, O. Diaz-Parra and J. C. Zavala-Diaz, International Journal of Combinatorial Optimization Problems and Informatics, **2**, 25-31 (2011).
[11] K. R. Baker, Principles of Sequencing and Scheduling, Wiley Publishing, (2009).
[12] T. Yamada and R. Nakano, Parallel Problem Solving from Nature, **2**, 281-290 (1992).
[13] M. Kammer, M. V. D. Akker and H. Hoogeveen, Computers and Operations Research, **38**, 1556-1561 (2011).
[14] T. F. Abdelmaguid, Journal of Software Engineering and Applications, **3**, 1155-1162 (2010).
[15] A. Sprecher, R. Kolisch and A. Drexel, European Journal of Operational Research, **80**, 94-102 (1995).

[16] B. Giffler and G. L. Thompson, Operations Research, **8**, 487-503 (1960).
[17] Lee H. P. and S. Sutinah, MATEMATIKA, **22**, 91-107 (2006).

Kim Jun Woo received the Master's degree in industrial engineering from Korean Advanced Institute of Science and Technology in 2003, and the Ph.D in industrial systems and management engineering from Korean Advanced Institute of Science and Technology in 2009. He is currently an assistant professor of the department of the industrial and management systems engineering, Dong-A University. His current research interests include intelligent systems, data mining, meta-heuristics, serious games and service science, etc.