Flow-shop and Job-shop Scheduling Problems Solved by Simulation Models

Nuntapon Dilokcharoen, Nattapoom Charoenlarpkul, Pisut Pongchairerks

Faculty of Engineering, Thai-Nichi Institute of Technology
Bangkok, Thailand
[1]di.nuntapon_st@tni.ac.th
[2]ch.nattapoom_st@tni.ac.th
[3]pisut@tni.ac.th

Abstract— ARENA is the most famous simulation software in Thailand; industrial engineers usually know how to use this software. This paper thus attempts to develop simulation models for flow-shop scheduling problems and job-shop scheduling problems via the ARENA software product. The proposed simulation models are so simple and use a small CPU memory space so that they can be applied in a non-commercial version of ARENA for solving large-scale job-shop scheduling and flow-shop scheduling problems.

Keywords— ARENA; Simulation; Job-shop scheduling problem; Flow-shop scheduling problem; JSP; FSP

I. INTRODUCTION

Nowadays, the readymade scheduling software products in the market are very expensive and may not be adjusted for a specific scheduling problem that can be found only in a particular factory. In order to help small and medium enterprises (SMEs) which may not have strong financial situations to have a well-performing scheduling software solution, this paper tries to develop simulation models for two frequently found scheduling problem, i.e. flow-shop scheduling problem and job-shop scheduling problem. To do so, this paper uses ARENA software product as the framework, because ARENA is very famous in Thailand such that many companies already have this software product and its price is not expensive. ARENA is a discrete-event simulation program designed for analyzing the impact of changes in many applications such as supply chain and logistics, manufacturing processes, distribution and warehousing, and service systems. As just mentioned, this software product uses a discrete-event simulation so that there is no change in system assumed to occur between two consecutive events.

Flow-shop scheduling problem and job-shop scheduling problem both are hard-to-solve scheduling problem to make a sequence to operate a number of jobs on a number of machines. Job-shop scheduling problem (JSP) and flow-shop scheduling problem (FSP) both comes with a set of given jobs and a set of given machines. Each job consists of a number of operations which must be processed in a predefined sequence. In JSP, the sequence of operations of a job may or may not be the same to the sequence of operations of another job; on the contrary, in FSP, the sequence of operations of every job is exactly same.

The contribution of this paper is to develop simple and cheap simulation models for JSP and FSP for SMEs that do not have enough financial supports to purchase a readymade scheduling software product. This paper is divided into five sections which have their topics as follows: a review of the literature related to the research written in this paper, simulation models for FSP and JSP via ARENA, tests of the performances of the proposed ARENA simulation models, and a conclusion of the findings given by this paper.

II. LITERATURE REVIEW

A. ARENA Software and Its Applications

ARENA, as summarized in Takus and Profozich (1997), is a flexible and powerful software product which can create animated simulation models representing the real systems being studied. The software product uses graphical objects, called modules, so as to define system components, e.g. jobs, machines, operators, etc. ARENA is coded based on SIMAN simulation language; thus the user is allowed to directly write SIMAN code in order to increase the flexibility for ARENA software. ARENA is now installed in more than 1,500 sites worldwide. It has been applied in many systems, e.g. automotive factory, electronics factory, etc. For example, Liong and Loo (2009) use ARENA to make a simulation study of warehouse loading and unloading systems. In Dehaghi et al. (2012), ARENA is used to determine the optimal order quantities in lot sizing models. In Goldsman et al. (2002), ARENA is applied for a transportation logistics problem. It is also found that Nawara and Hassanein (2013) and Sousa and Moreira (2007) apply ARENA for solving job-shop scheduling problems; however, the ARENA simulation models of these two previously published papers are different from the ARENA simulation models proposed in this paper.

B. Statements of FSP and JSP

This paper considers on two major shop scheduling problems, i.e. flow-shop scheduling problem and job-shop scheduling problem. The flow-shop schedule is usually used when the product demand is high while, on the contrary, the job-shop schedule is usually used when the product demand is low or medium. The statements of these two shop scheduling problems are given below.

Flow-shop scheduling problem (FSP) starts with a set of n jobs J_i (i = 1,…, n) and a set of m machines M_j (j = 1,…, m). Each job J_i consists of an ordered set of m operations O_{i1}, O_{i2},…, O_{im}, and each operation requires to process on a different machine. In FSP, every job must be processed on all machines in the same order; however, the processing time on a same machine of different jobs may not be the same. The processing time of the operation O_{ij} is given by t_{ij}. The objective is to find a feasible allocation of all operations to time intervals on the given machines such that makespan is minimized. Hereafter, the allocation of all operations to time interval is called schedule. The makespan is the completion time of the last operation in the schedule. The constraints of FSP are the operations are not preemptable, each machine can handle only one job at any given time period and each job can be processed on only one machine at one given time period.

Job-shop scheduling problem (JSP) starts with a set of n jobs J_i (i = 1,…, n) and a set of m machines M_j (j = 1,…, m). Each job Ji consists of an ordered set of m operations O_{i1}, O_{i2},…, O_{im}. The given order of operations is predefined and cannot be change. Each operation O_{ij}must be processed by one given machine during t_{ij} time units. The objective is to find a feasible schedule in order to minimize makespan. Similar to FSP, the constraints of JSP are the operations are not preemptable, each machine can handle only one job at any given time period and each job can be processed on only one machine at one given time period. However, JSP is different from FSP in that the sequence of operations of a job may not be the same to the sequence of operations of another job. JSP is one of the most popular scheduling problems; so that many papers attempt to solve JSP such as Pongchairerks and Kachitvichyanukul (2009), Pongchairerks (2014), Pongchairerks (2014). The details of FSP and JSP are also found in Baker and Trietsch (2009) and Gen and Cheng (1997).

III. Simulation Models for FSP and JSP

This section presents the steps of developing the simulation models proposed in this paper in order to solve FSP and JSP problems. There are 12 steps of developing the proposed simulation model for FSP and 19 steps for developing the proposed simulation model for JSP.

A. Steps to Develop Simulation Model for FSP

The 12 steps of developing the proposed simulation model for FSP are shown below. Every solution given by this model is a non-delay schedule.

Step 1: Drag and drop a Create module named “Create 1” into the model window. Assign Entities per arrival equals to the number of given jobs (n), and assign Max Arrival`s equals 1. For example, if the number of all given jobs is 3, then Entities per arrival = 3. Since Max Arrivals = 1, all these three jobs will arrive just once in the same time. Here there is only one entity type containing n entities; each entity number represents a job.

Step 2: Create a text file named “ProcessTime” containing the processing times on machines for the given jobs. The number of rows equals to n while the number of columns equals to m. In the first row, the first column provides the processing time of O_{11}, the second column provides the processing time of O_{12}, and so on. In the second row, the first column provides the processing time of O_{21}, the second column provides the processing time of O_{22}, and so on. That means the i-th row in the text file provides the processing times of O_{i1}, O_{i2},…, O_{im}. Then assign this text file as an ARENA file by clicking the File spreadsheet in Advanced Process in Project Bar; clicking the browse button of the Operating System File Name and selecting the ProcessTime text file; naming this generated ARENA file as “ProcessTime”. Select Initialize Option = Close.

Step 3: Drag and drop a ReadWrite module named “ReadWrite 1” into the model window and connect it to Create 1. Select Arena File Name = ProcessTime; then add m attributes. Attribute 1, read from the first column of the ProcessTime file, contains the processing time of the first operation for every given job; Attribute 2, read from the second column of the ProcessTine file, contains the processing time of the second operation for every given; and so on. Note that the entity serial number 1 means Job 1, entity serial number 2 means Job 2.

Step 4: Drag and drop a Delay module named “Delay 1” into the model window and connect it to ReadWrite 1. In the Delay 1 module, assign the delay time to UNIF(0, 1). The Delay 1 module is used to make a random sequence for the given jobs.

Step 5: Drag and drop a Record module named “Record 1” into the model window and connect it to Delay 1. Set Type = Expression, Value = TNOW, Tally Name = Record 1. The Record 1 is used to record the starting times of all given jobs.

Step 6: Drag and drop a Process module named “Process 1” into the model window and connect it to Record 1. In Process 1, assign Action = Seize Delay Release, Delay Type = Expression, and Expression = Attribute 1, then add a resource named “Resource 1” with a quantity of 1. The process 1 module represents the first operation of every job. Then drag and drop a Process module named “Process 2” into the model window and connect it to Process 1. In Process 2, assign

Action = Seize Delay Release, Delay Type = Expression, and Expression = Attribute 2, then add a resource named "Resource 2" with a quantity of 1. The process 2 module represents the second operation of every job. Repeat the same procedure for the rest operation. For example, if every job has 3 operations, the simulation model must have 3 Process modules, i.e. Process 1, Process 2, and Process 3.

Step 7: Drag and drop a Record module named "Record 2" into the model window and connect it to the last process module; select Type = Count, Value = 1, and Counter Name = Record 2.

Step 8: Drag and drop a Record module named "Record 3" into the model window and connect it to the last Process module. In Record 3, set Type = Expression, Value = TNOW, and Tally Name = Record 3. The Record 3 is used to record the finished times of all given jobs.

Step 9: Drag and drop a Record module named "Record 4" into the model window and connect it to Record 3. In Record 4, set Type = Expression, Value = TMAX(Record 3) – TMIN(Record 1), and Tally Name = Makespan. The Record 3 is used to calculate the makespan values of all given jobs.

Step 10: Drag and drop a Record module named "Record 5" into the model window and connect it to Record 4. Then, select Type = Expression, Value = Entity.SerialNumber; click Record into Set, and set Tally Set Name = Tally Set 1, and Set Index = NC(Record 4).

Step 11: Drag and drop a Dispose module named "Dispose 1" into the model window and connect it to Record 5.

Step 12: Save the completed file in the name of "FSP.doe". (User may use other names.)

Figure 1 shows the model structure for *n*-jobs/3-machines FSP instances as an example. This model structure can be used for the JSP instances of three machines but every number of jobs. The simulation model for other n-jobs/m-machines FSP instances can be constructed by following the 12 steps above given.

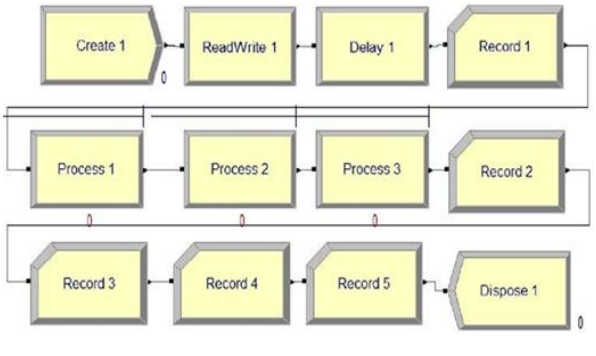

Fig. 1 Proposed FSP model for *n*-jobs/3-machines instances

B. Steps to Develop Simulation Model for JSP

The 18 steps of developing the proposed simulation model for job-shop scheduling problem are given below. Every solution given by this model is a non-delay schedule.

Step 1: Drag and drop *n* Create modules, namely "Create 1", "Create 2",…, "Create *n*", into the model window. For example, if the JSP problem has 3 jobs, the model must have 3 Create modules. For each Create module, assignEntities per Arrival equal to 1, and assign Max Arrivals equal 1. Here there are *n* entity types, each of which has only one entity; each entity thus represents a job.

Step 2: Create text files named "MachineSequence 1", "MachineSequence 2",…, "MachineSequence *n*" containing the sequence of machines for the job J_1, the sequence of machines for the job J_2,…, the sequence of machines for the job J_n, respectively.For each file, the number of rows equals to 1 while the number of columns equals to *m*. The first column in the file means the machine used for the first operation, the second column means the machine used for the second operation, and so on. Then assign these text files as ARENA file by clicking the File spreadsheet in Advanced Process in Project Bar. In the first row of the File spreadsheet, click the browse button of the Operating System File Name, select the MachineSequence 1 text file, name this generated ARENA file as "MachineSequence 1", and select Initialize Option = Close. In the second row of the File spreadsheet, click the browse button of the Operating System File Name, select the MachineSequence 2 text file, name this generated ARENA file as "MachineSequence 2", and select Initialize Option = Close. Repeat the same procedure for the MachineSequence 3 text file through the MachineSequence *n* text file.

Step 3: Create a text file named "ProcessTime" containing the processing times on machines for the given jobs. The number of rows equals to *n* while the number of columns equals to *m*. In the first row, the first column provides the processing time of O_{11} on the machine given in the MachineSequence file, the second column provides the processing time of O_{12} on the machine given in the MachineSequence file, and so on. In the second row, the first column provides the processing time of O_{21} on the machine given in the MachineSequence file, the second column provides the processing time of O_{22} on the machine given in the MachineSequence file and so on. That means the *i*-th row in the text file provides the processing times of O_{i1}, O_{i2},…, O_{im}. Then assign this text file as an ARENA file by clicking the File spreadsheet in Advanced Process in Project Bar; clicking the browse button of the Operating System File Name and selecting the ProcessTime text file; naming this generated ARENA file as "ProcessTime". Select Initialize Option = Close.

Step 4: Drag and drop n ReadWrite modules named "ReadWrite 1", "ReadWrite 2",..., "ReadWrite n", into the model window. Connect the ReadWrite 1 module to the Create 1 module, connect the ReadWrite 2 module to the Create 2 module, and so on. In the ReadWrite i module (i = 1,…, n), select Arena File Name = MachineSequence$i$,and add m attributes into Assignments; Attribute 1 shown in the first column of the MachineSequence $i$file contains the machine number of the first operation of the job J_i, Attribute 2 shown in the second column of the MachineSequence $i$file contains the machine number of the second operation of the job J_i, and so on.

Step 5: Drag and drop a ReadWrite module named "ReadWrite n+1" into the model window and connect it to ReadWrite 1, ReadWrite 2,…, ReadWrite $n$modules. In ReadWrite n+1 module, select Arena File Name = ProcessTime, and add$m$ attributes into Assignments, namely "Attribute 1.Time",…, "Attribute m.Time". Attribute 1.Time shown in the first column of the ProcessTime file contains the processing time of the first operation for every given job, Attribute 2.Time shown in the second column of the ProcessTime file contains the processing time of the second operation for every given job, and so on.

Step 6: Drag and drop a Delay module named "Delay 1" into the model window and connect it to ReadWrite n+1. In the Delay 1 module, assign the delay time to UNIF(0, 1). The Delay 1 module is used to make a random sequence for the given jobs.

Step 7: Drag and drop an Assign module named "AssignProcessCounter" into the model window and connect it to Delay 1. Then, add one attribute named "Round" into Assignments; let Type = Attribute, and New Value = 1.

Step 8: Drag and drop a Record module named "Record 1" into the model window and connect it to the AssignProcessCounter module. Set Type = Expression, Value = TNOW, Tally Name = Record 1. The Record 1 is used to record the starting times of all given jobs.

Step 9: Drag and drop a Record module named "Record 2" into the model window and connect it to the Record 1 module. Then set Type = Count, Value = 1, and Counter Name = Record 2.

Step 10: Drag and drop a Record module named "Record 3" into the model window and connect it to the Record 2 module. Then set Type = Expression, Value = Entity.SerialNumber, and click Record into Set. Then set Tally Set Name = Tally Set 1, and Set Index = NC(Record 2).

Step 11: Drag and drop a Decide module named "DecideStep" into the model window and connect it to Record 3. Set Type = N-way by Condition; and add m conditions. The first condition is whether the attribute named "Round" = 1 or not, i.e. set If = Attribute, Name = Round, Is = '==', and Value = 1; the second condition is whether the attribute named "Round" = 2 or not, i.e. set If = Attribute, Name = Round, Is = '==', and Value = 2; and so on, until the m-th condition as whether the attribute named "Round" = m or not, i.e. set If = Attribute, Name = Round, Is = '==', and Value = m. This module is to assign a particular job to a machine for operating the earliest unprocessed operation. If the value of the Round attribute of a particular job equals 1, it means this job will be assign to a machine corresponding to the first operation; if the value of the Round attribute of a particular job equals 2, it means this job will be assigned to a machine corresponding to the second operation; and so on.

Step 12: Drag and drop m Assign modules, named "AssignProcessTime 1", "AssignProcessTime 2",…, and "AssignProcessTime$m$" into the model window. Connect the AssignProcessTime 1 module to the first condition of the DecideStep module, connect the AssignProcessTime 2 module to the second condition of the DecideStep module, and so on. In the AssignProcessTime 1 module, add an attribute named "ProcessTime" into Assignments by setting Type = Attribute, Attribute Name = ProcessTime, and New Value = Attribute 1.Time; in the AssignProcessTime 2 module, add an attribute named "ProcessTime" into Assignments by setting Type = Attribute, Attribute Name = ProcessTime, and New Value = Attribute 2.Time; and so on.

Step 13: Drag and drop m Decide modules, named "Decide 1", "Decide 2",…, and "Decide m" into the model window. Connect the Decide 1 module to the AssignProcessTime 1 module, the Decide 2 module to the AssignProcesTime 2 module, and so on. In the Decide 1 module, assign Type = N-way by Condition, and create m conditions; the j-th condition is created by setting If = Attribute, Name = Attribute 1, Is = '==', and Value = j–1; where j = 1,…, m. In the Decide 2 module, assign Type = N-way by Condition, and create m conditions; the j-th condition is created by setting If = Attribute, Name = Attribute 2, Is = '==', and Value = j–1. Setting the values in the remaining Decide modules using the same procedure, excepting the name of attribute in conditions, i.e. Name = Attribute 3 for the Decide 3 module,Name = Attribute 4 for the Decide 4 module, and so on.

Step 14: Drag and drop m Process modules, named "Machine 1", "Machine 2",…, and "Machine m" into the model window. Connect the first conditions of all m Decide modules to the Machine 1 module, the second conditions of all m Decide modules to the Machine 2 module,…, and the m-th conditions of all m Decide modules to the Machine m module. In the Machine 1 module, set Action = Seize Delay Release, add one resource named "Resource 1" with quantity = 1, set Delay Type = Expression, and Expression = ProcessTime; in the Machine 2 module, set Action = Seize Delay Release, and add one resource named

"Resource 2" with quantity = 1, set Delay Type = Expression, and Expression = ProcessTime; and so on.

Step 15: Drag and drop an Assign module named "RoundCount" and connect it to all *m* Process modules. In this module, add one assignment by set Type = Attribute, Attribute Name = Round, New Value = Round+1. Then connect the back side of the RoundCount module to the front side of the DecideStep" module.

Step 16: Drag and drop a Record named "Record 4" and connect it to the Else condition of the DecideStep module. Then set Type = Expression, Value = TNOW, and Tally Name = Record 4.

Step 17: Drag and drop a Record named "Record 5" and connect it to the Recode 4 module. Then set Type = Expression, Value = TMAX(Record 3) – TMIN(Record 1), and Tally Name = Record 5.

Step 18: Drag and drop a Dispose module named "Dispose 1" into the model window and connect it to Record 5.

Step 19: Save the completed file in the name of "JSP.doe". (User may use other names.)

Figure 2 shows the model structure for 3-jobs/3-machines JSP instances as an example. This model structure can be used for the JSP instances of three machines but every number of jobs. The simulation model for other *n*-jobs/*m*-machines JSP instances can be constructed by following the 19 steps above given.

After the FSP or JSP model has been run successfully, the output data will be shown in the file FSP.out for FSP model or JSP.out for JSP model. In the output file, the sequence of jobs putted into the system as a solution given by a replication of the model is shown in the table name "Tally Variables". In this table, the value of the Tally 1 is the first job putted into the system, the value of the Tally 2 is the second job putted into the system; Replication ended at time is the makespan value of the solution given. For example, the summary for replication 1 of the FSP.out of the proposed FSP model in car01 instance is shown in Figure 3. In Figure 3, the solution is the schedule that processes all 11 given jobs in this following order: J_3, J_5, J_4, J_{10}, J_2, J_1, J_7, J_{11}, J_9, J_8 and J_6; the makespan value of this solution is 8707.

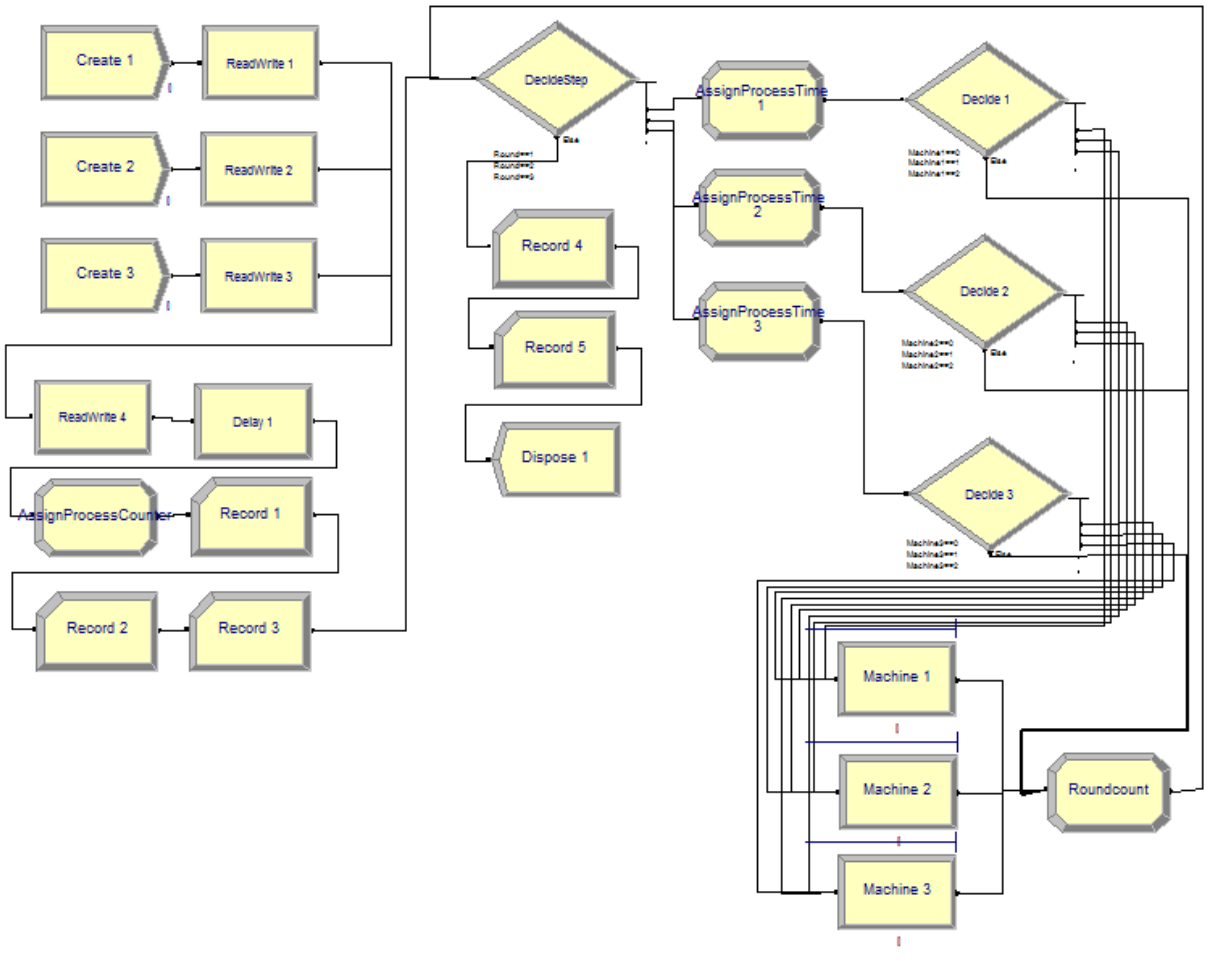

Fig. 2 Proposed JSP model for 3-jobs/3-machines instances

ARENA Simulation Results
MSI - License: 2957005515

Summary for Replication 1 of 10000

Project: Unnamed Project
Analyst: Nuntapon

Run execution date : 9/18/2014
Model revision date: 9/18/2014

Replication ended at time : 8707.0 Seconds
Base Time Units: Seconds

TALLY VARIABLES

Identifier	Average	Half Width	Minimum	Maximum	Observations
Tally 1	3.0000	(Insuf)	3.0000	3.0000	1
Tally 2	5.0000	(Insuf)	5.0000	5.0000	1
Tally 3	4.0000	(Insuf)	4.0000	4.0000	1
Tally 4	10.000	(Insuf)	10.000	10.000	1
Tally 5	2.0000	(Insuf)	2.0000	2.0000	1
Tally 6	1.0000	(Insuf)	1.0000	1.0000	1
Tally 7	7.0000	(Insuf)	7.0000	7.0000	1
Tally 8	11.000	(Insuf)	11.000	11.000	1
Tally 9	9.0000	(Insuf)	9.0000	9.0000	1
Tally 10	8.0000	(Insuf)	8.0000	8.0000	1
Tally 11	6.0000	(Insuf)	6.0000	6.0000	1

Fig. 3 The solution of car01 instance given from the first replication of the proposed FSP model

IV. Experiment and Results

To evaluate the performances of the proposed models, this paper uses the proposed FSP model on five benchmark instances, i.e. car1 (11x5), car3 (12x5), car5 (10x6), rec1 (20x7) and rec7 (20x10). The parentheses after the instance's name show the number of jobs and the number of machines. The results given by the proposed FSP model are compared to the results given by Davendra and Onwubola (2007). This paper then uses the proposed JSP model on ten benchmark instances, i.e. abz5 (10x10), abz6 (10x10), la16 – la20 (10x10), and la21 – la23 (15x10). The results given by the proposed JSP model are compared to the results given by Asadzadeh (2014). The input parameters when setting up of the proposed models on this experiment are given as follows: In run setup, let number of replications = 10,000, warm-up period = 0, replication length = infinite, base time units = second. After that, the FSP or JSP model is ready to run. Note that each replication gives one single solution. Thus, as this experiment uses 10,000 replications, ARENA will return 10,000 solutions. The value of the best solution given from all 10,000 replications of the proposed FSP model will be compared to the best found solution value taken from Davendra and Onwubola (2007) for FSP and that of the proposed JSP model will be compared to the best solution value found from Asadzadeh (2014) for JSP.

Table I compares the value of the best found solution of Davendra and Onwubola (2007) and the value of the best found solution of the FSP model proposed in this paper for each instance. It also shows the average of the 10,000 solution values found from the FSP model and the computational time used per replication. Let BFS means the value of the best found solution and AFS means the average of all solution values.

TABLE I
RESULTS FROM FSP MODEL

Instance	BFS from Davendra and Onwubola	BFS from the Proposed FSP Model	AFS from the Proposed FSP Model	CPU Time per Replication (sec.)
car1	7038	7279	8355	0.082
car3	7312	7789	8441	0.034
car5	7720	8490	8977	0.019
rec1	1247	1394	1554	0.028
rec7	1566	1669	1922	0.075

Table II then compares the value of the best found solution of Asadzadeh (2014) and the value of the best found solution of the proposed JSP model proposed for each instance.

TABLE II
RESULTS FROM JSP MODEL

Instance	BFS from Asadzadeh	BFS from the Proposed JSP Model	AFS from the Proposed JSP Model	CPU Time per Replication (sec.)
abz5	1234	1351	1378	0.044
abz6	943	1014	1053	0.020
la16	945	1070	1099	0.005
la17	784	897	903	0.003
la18	848	933	942	0.007
la19	842	968	988	0.005
la20	902	1052	1078	0.006
la21	1046	1196	1298	0.004
la22	927	1106	1204	0.087
la23	1032	1128	1157	0.005

Based on the results given, the proposed simulation models perform very well in terms of computational speed, but not quite well in terms of solution quality when comparing them to the state-of-the-art algorithms. The best found solution value of the proposed FSP model is worse than the best found solution value of Davendra and Onwubola (2007) around 7.7% and the best found solution value of the proposed JSP model is worse than that of Asadzadeh (2014) around 11.4% on average. The results thus show that these proposed models need some heuristics to enhance their performances for the next researches.

V. CONCLUSIONS

This paper proposes the ARENA simulation models for FSP and JSP. These models can use with the ARENA's student version for solving large scale FSP or JSP instances; therefore, companies can develop a free software product by themselves using the models proposed in this paper for solving FSP and JSP problems. The models also consume very short computational times. Although the models cannot compare to the state-of-the-art algorithms in terms of solution quality, they are very worthwhile in terms of cost and flexibility. The further work of this research is to add an algorithm into the simulation models in order to enhance the performances of the proposed models.

REFERENCES

[1] Asadzadeh, L. 2014. Solving the job shop scheduling problem with a parallel and agent-based local search genetic algorithm. Journal of Theoretical and Applied Information Technology, 62: 317-324.
[2] Baker, K.R. and Trietsch, D. 2009. Principles of Sequencing and Scheduling, John Wiley & Sons.
[3] Davendra, D. and Onwubolu, G. 2007. Flow shop scheduling using enhanced differential evolution algorithm. Proceedings of the 21st European Conference on Modelling and Simulation, Prague, Czech Republic, pp: 259-264.
[4] Dehaghi, Z.H., Sajadi, S.M. and Ahmadabadi, M.N. 2012. Determine the optimal order quantities in lot sizing models regarding minimum order quantity with simulation. International Journal of Engineering Sciences, 2: 399-403.
[5] Gen, M. and Cheng, R. 1997. Genetic Algorithms and Engineering Design, John Wiley & Sons.
[6] (2002) The IEEE website. [Online]. Available: http://www.ieee.org/
[7] Goldsman, D., Pernet, S. and Kang, K. 2002. Simulation of transportation logistics. Proceedings of the 2002 Winter Simulation Conference, Atlanta, GA, pp: 901-904.
[8] Liong, C.-Y. and Loo, C.S.E. 2009. A simulation study of warehouse loading and unloading systems using ARENA. Journal of Quality Measurement and Analysis, 5: 45-56.
[9] Nawara, G.M. and Hassanein, W.S. 2013. Solving the job-shop scheduling problem by Arena simulation software. International Journal of Engineering Innovation and Research, 2: 161-166.
[10] Sousa, P.S.A. and Moreira, M.R.A. 2007. Performance analysis of job-shop production systems under different order release control parameters. Proceedings of the World Congress on Engineering 2007, London, UK, pp: 1056-1061.
[11] Pongchairerks, P. and Kachitvichyanukul, V. 2009. A two-level particle swarm optimization algorithm on job-shop scheduling problems. International Journal of Operational Research, 4: 390–411.
[12] Pongchairerks, P. 2009. Particle swarm optimization algorithm applied to scheduling problems. Scienceasia, 35: 89-94.
[13] Pongchairerks, P. 2014. A self-tuning PSO for job-shop scheduling problems. International Journal of Operational Research, 19: 96-113.