A GENETIC PROGRAMMING BASED HYPER-HEURISTIC FOR PRODUCTION SCHEDULING IN APPAREL INDUSTRY

Cecilia E. Nugraheni[1], Luciana Abednego[1], and Maria Widyarini[2]

[1]Dept. of Computer Science, Parahyangan Catholic University, Bandung, Indonesia
[2]Dept. of Business Adm., Parahyangan Catholic University, Bandung, Indonesia

ABSTRACT

The apparel industry is a type of textile industry. One of scheduling problems found in the apparel industry production can be classified as Flow Shop Scheduling Problems (FSSP). GPHH for FSSP is a genetic programming based hyper-heuristic techniques to solve FSSP[1]. The algorithm basically aims to generate new heuristics from two basic (low-level) heuristics, namely Palmer Algorithm and Gupta Algorithm. This paper describes the implementation of the GPHH algorithm and the results of experiments conducted to determine the performance of the proposed algorithm. The experimental results show that the proposed algorithm is promising, has better performance than Palmer Algorithm and Gupta Algorithm.

KEYWORDS

Hyper-heuristic, Genetic Programming, Palmer Algorithm, Gupta Algorithm, Flow Shop Scheduling Problem, Apparel Industry

1. INTRODUCTION

Generally, the textile industry can be divided into two major segments, namely the textile industry and the apparel industry [2,3]. The production of fabric from raw material is done through a series of processes such as spinning, weaving, knitting, etc. These processes are the focus of the textile industry. Meanwhile, the apparel industry transforms fabrics into ready-to-use goods, in particular ready-to-wear clothes. The activities that belong to the apparel industry are pattern making, cutting, sewing, and finishing.

Based on the flow of work activities, the production system of the apparel industry is usually included in the flow shop production or job shop production group. Job Shop is a type of production process flow that is used for producing goods with small production quantities but have many models or variants. "Custom-made" products that have to follow the unique design and special specifications of the customer at a specified time and cost usually use this type of production process flow. Flow Shop Production is a type of production process that is used to produce products that are assembled or produced in large quantities and successively (continuous). All products are manufactured with the same standards and processes.

This work focuses on Flow Shop Scheduling Problems (FSSP) which is a class of scheduling problems found in the manufacturing industry whose workflow follows the Flow Shop Production. Given a number of jobs that must be processed in a series of stages, the goal of FSSP

David C. Wyld et al. (Eds): MLNLP, BDIoT, ITCCMA, CSITY, DTMN, AIFZ, SIGPRO - 2020
pp. 119-128, 2020. CS & IT - CSCP 2020 DOI: 10.5121/csit.2020.101212

is to find a sequence of jobs that meets certain optimal criteria. There are many approaches proposed for these problems that are based on heuristic techniques. There are three types of heuristics. The first type is called low-level heuristics such as standard dispatching rules (First Comes First Served, Shortest Processing Time, etc.) and several algorithms such as NEH, Palmer, Gupta, Dannenbring, Pour, etc.[4,5,6], The second type is meta-heuristics such as Genetic Algorithm, Simulated Annealing, Ant Colony Algorithm, etc. [7,8]. The last type is hyper-heuristics such as Genetic Programming [9,10,11]. Different from the other heuristics, hyper-heuristic does not work directly on the problem domains, rather on the heuristics. This is why hyper-heuristic is usually called heuristics to choose heuristics. This characteristic enables hyper-heuristic to do the searching in a more flexible way. Also, hyper-heuristics offers the ease of application to a larger scope of problems.

In our previous work [1], we proposed a technique for solving FSSP problems which is a hyper-heuristic based genetic programming. The main idea of the technique is to generate new low-level heuristics by combining two low-level heuristics namely Palmer Algorithm and Gupta Algorithm with the use of the Genetic Programming technique. The combination of two algorithms has been not reported before. Continuing the work, we have developed a program that implements the proposed technique. This paper reports the implementation as well as the results of experiments conducted for measuring its performance.

The rest of this paper is organized as follows. Section 2 briefly describes the GPHH algorithm that we proposed to solve FSSP problems. We briefly describe what FSSP is, the types of heuristics used to solve the FSSP and the algorithm we proposed. A more complete explanation of this can be found in [1]. Section 3 explains the implementation of GPHH. Section 4 presents the experimental results. Last, conclusion and future work are given in Section 5.

2. PROPOSED TECHNIQUE

Given m machines and n jobs to be processed on each machine, the objective of the FSSP is to find job sequences that meet certain optimality objectives. In this study, the objective used is makespan, which is the time needed to process the entire job starting from the first job in the first machine to the final time of processing the last job on the last machine. It is assumed that at each stage there is only one machine provided for processing the jobs.

Palmer's algorithm and Gupta's algorithm have very similar working principles. In order to generate job orders, both algorithms use a value that is known as the slope index. The two algorithms consist of two stages, namely calculating the slope index of each job and sort the jobs according to the slope indices.

Genetic programming uses the same idea as the genetic algorithm. It is inspired by Darwin's theory of evolution. The difference between genetic programming and genetic algorithm lies in the role of the chromosomes. In genetic programming, a chromosome represents a computer program (function) that can be used to find solutions to problems. The application of genetic operators (crossover, mutation, and reproduction) generates new computer programs. Figure 1 gives two examples of computer programs represented as syntax trees (p-2 and p/1+g), as well as the results of the crossover (p/1-2 and p+g). For mutation operation, a randomly generated sub-tree will replace the part of original tree based on a mutation point. Figure 2 shows an example of mutation operation over computer program. The replacement of constant 2 by p+g resulting in a new computer program which is (p/1 – (p + g)).

The algorithm of GPHH is given in Figure 3. Very similar to genetic algorithms, the algorithm starts with an initial population consisting of some individuals representing a set of computer

programs. Then, iteratively, a new population is generated by applying some genetic operations. Given a set of inputs namely a set of operands (constants and variables), a set of operators, population size, pool size, maximum depth of the syntax tree, the number of the run, the number of generation, FSSP problem, crossover rate, and mutation rate, this algorithm returns the schedule or the order of the jobs and the makespan corresponding to it.

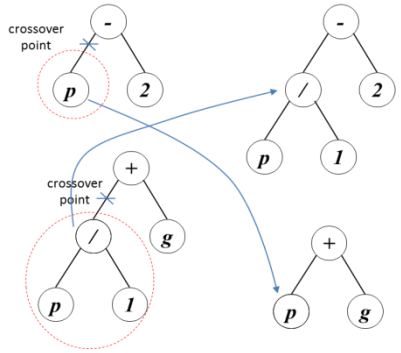

Figure 1. An example of crossover operation.

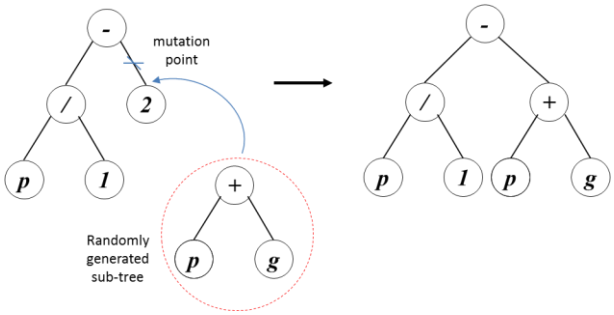

Figure 2. An example of mutation operation.

function GPHH (*ops*, *oprs*, *popSize*, *poolSize*, *depth*, *maxRun*, *maxGen*, *problem*, *COrate*, *Mrate*) → (schedule, makespan)

1. Generate a pool of computer programs based on *ops* and *oprs* which are a set of operands and a set of operators, respectively. Two parameters are used in this stage, namely *depth* representing the depth of programs' syntax trees and *poolSize* representing the size of the pool or the number of generated programs.

2. **for** i = 1 **to** *maxRun* **do**

a. Create the initial population with the size of the population is *popSize* by randomly picking computer programs in the program pool.

b. **for** j = 1 **to** *maxGen* **do**

i. Generate a new population by applying genetic operations over computer programs in the current population and using the COrate and Mrate as the rate of cross over and mutation operation, respectively.

ii. Apply fitness measure to individuals in the new population.

iii. Record the best individual so far as well as the schedule and the makespan corresponding to it.

iv. j = j +1

c. i = i + 1

3. **return** the schedule and the makespan of best individual.

Figure 3. GPHH Algorithm.

3. IMPLEMENTATION

We have developed a computer program implementing the algorithm of GPHH as described above. In the implementation we assumed that the set of the operands and the operators, ops and

oprs, are fixed, which are {p, g, 1, 2} and {+, -, *, /}, respectively. The symbol p is used to represent the slope index of Palmer Algorithm and the symbol g is used to represent the slope index of Gupta Algorithm.

The user interface of the programs is given in Figure 4. The InputFile button is used for choosing a file representing the problem to be solved. There are two tabs, namely Problem Tab and Solution Tab. The Problems tab displays the problem in a tabular form which describes the time required for each machine to process each job. The Solution Tab is used for displaying the schedule and makespan resulted by Palmer Algorithm, Gupta Algorithm, and GPHH as shown in Figure 4.

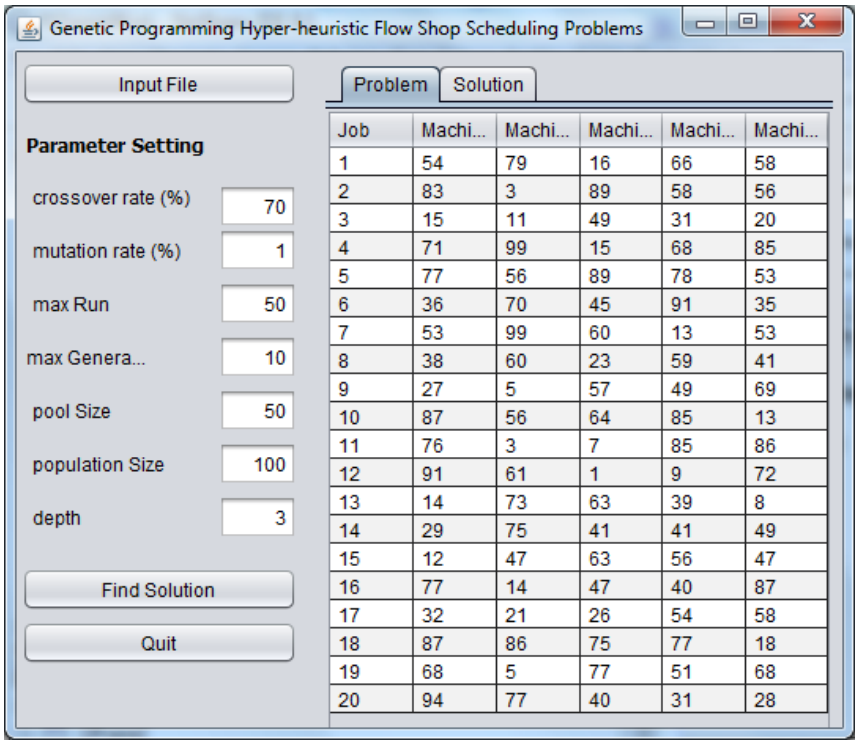

Figure 4. The problem tab for displaying the problem.

4. EXPERIMENTAL RESULTS AND ANALYSIS

In order to measure the performance of the heuristics generated by GPHH, a number of experiments were carried out. In carrying out the experiments, a benchmark proposed by Taillard et al. [12]. This benchmark provides a number of scheduling problems grouped by the number of jobs and machines as shown in Table 1. Each group consists of 10 problem instances. So in total there are 120 problem instances.

Experiments were carried out with different crossover rate COrate, namely 75%, 80%, and 85%. The rate of the mutation operation Mrate is set 5%. For each COrate we have used three depth values of syntax tree which are 2, 3, and 4. Meanwhile, the values for the other parameters are fixed, namely: maxRun = 50, maxGen = 10, poolSize = 250, and popSize = 200. The objective of the experiments is to compare the makespan generated by the GPHH algorithm with the makespan from Taillard's benchmarks, the makespans produced by the Palmer Algorithm and the makespans produced by Gupta Algorithm. Besides, the experiments also aim to know the effect of the crossover rate and the depth of syntax tree on the resulting makespan.

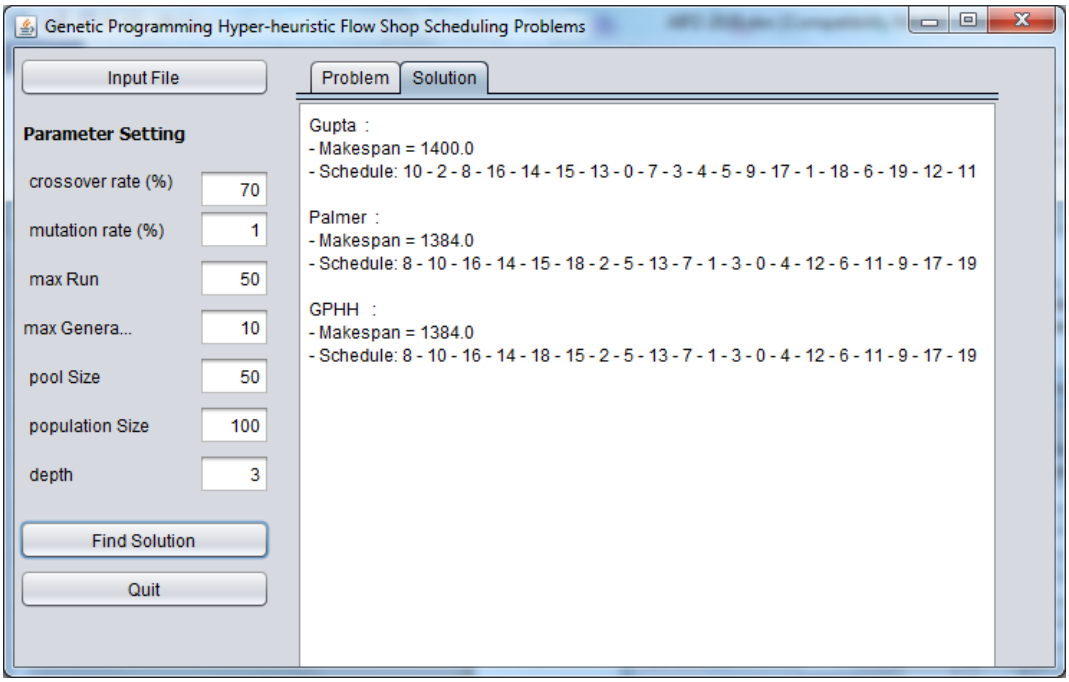

Figure 5. The solution tab for displaying the schedule and makespan of each algorithm.

Table 1. Problem group of Taillard Benchmark.

Group	The number of the jobs	The number of the machine
1	20	5
2	20	10
3	20	20
4	50	5
5	50	10
6	50	20
7	100	5
8	100	10
9	100	20
10	200	10
11	200	20
12	500	20

Figures 6, Figure 7, and Figure 8 compare the average makespans resulted from GPHH Algorithm with the corresponding makespans produced by other algorithms for each COrate value. For each syntax depth tree, depth, Figure 9 and Figure 10 show the comparison of average makespans of each instance problem group and the total average makespan.

From the experimental results, it can be concluded that although the performance of GPHH is still below the benchmark, in general, GPHH produces better makespan compared to Palmer Algorithm and Gupta Algorithm. The worst performance of GPHH occurs in the case of 500 jobs 20 machines. It can be seen that for the three experiments (Figure 6, 7, and 8), GPHH takes the last position. It can be observed, for the case of 500 job 20 machines, the benchmarks are also worse than Palmer and Gupta. Whether cases with many very large jobs will reduce the performance of GPHH still needs to be further analyzed and tested.

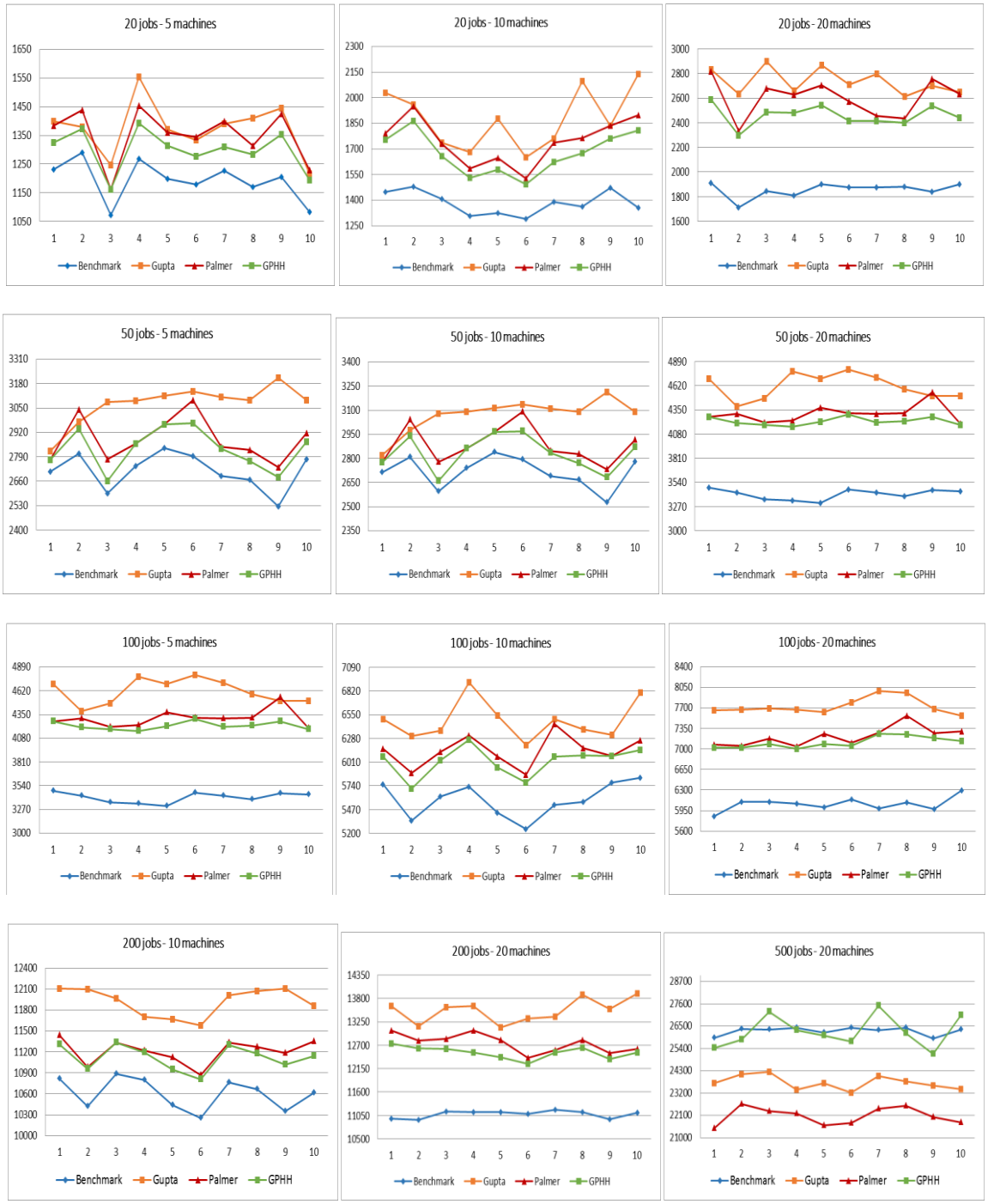

Figure 6. Experimental Results for Cross Over Rate 0.75.

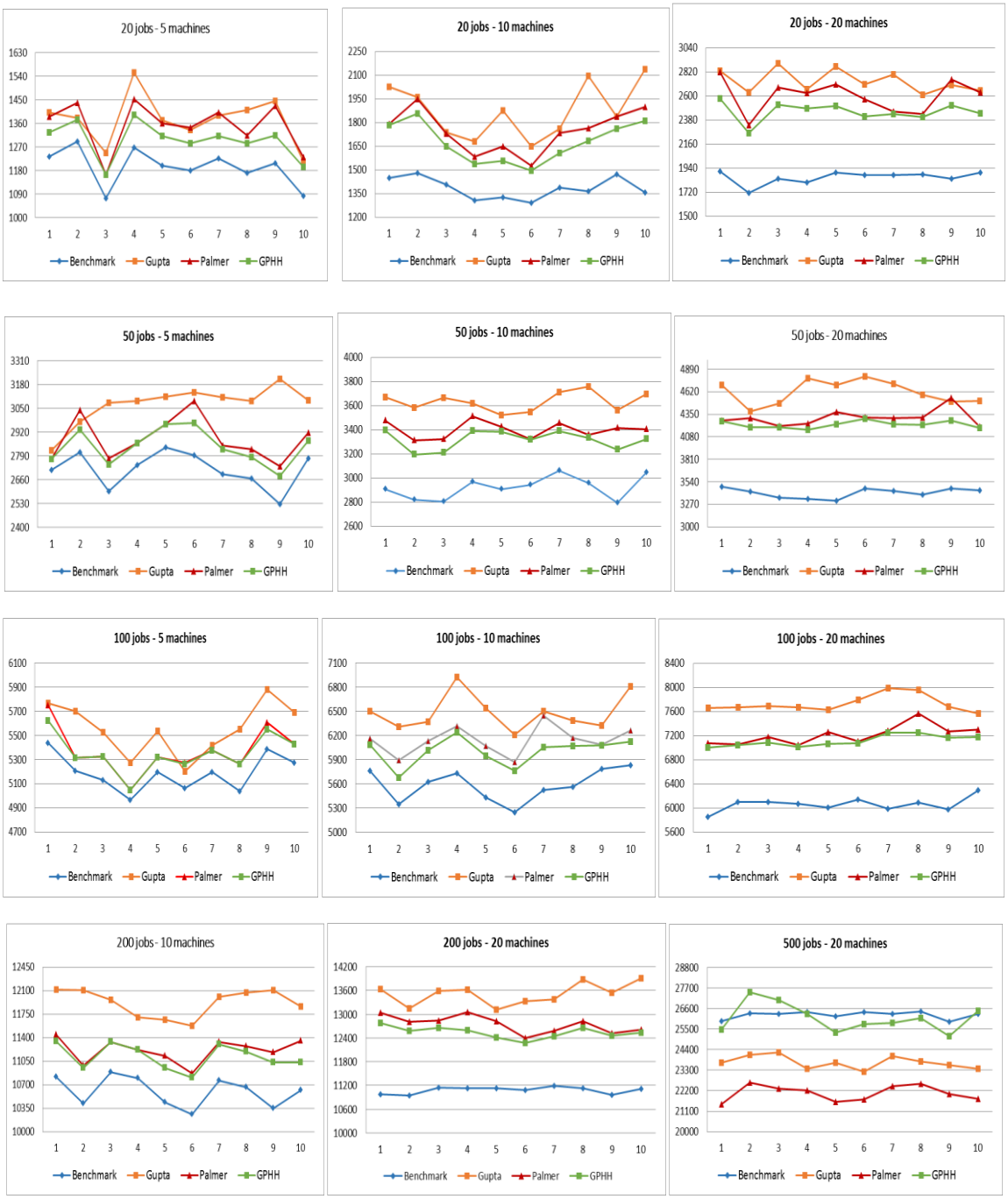

Figure 7. Experimental Results for Cross Over Rate 0.80.

Figure 8. Experimental Results for Cross Over Rate 0.85.

In terms of the depth of the syntax tree, from Figure 9 it can be seen that the three tree depth values show results that are not too different from one another for each problem instance group. However, from the total average makespan, the value 3 yields the best result compared to values 2 and 4 as shown in Figure 10.

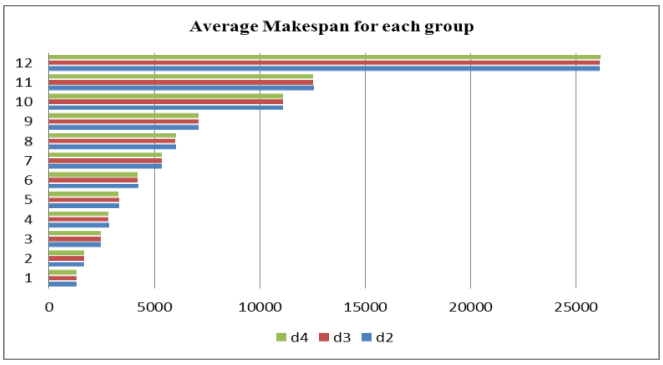

Figure 9. Average Makespans for each problem instance group.

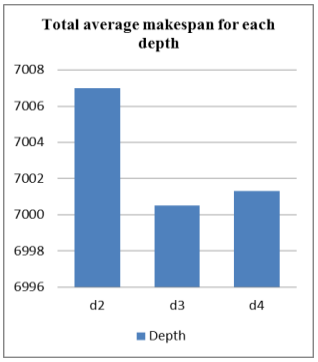

Figure 10. Total average for each syntax tree depth.

5. CONCLUSIONS

Scheduling problems in the textile industry in general belong to the Flow Shop Scheduling Problem (FSSP). Given a set of machines and a set of jobs, the objective of FSSP is to find a job order that meets some optimization criteria. In this work, a computer program that can be used to solve FSSP has been developed. This program implements the GPHH Algorithm which is genetic programming based hyper-heuristic for solving FSSP proposed in [1]. The experimental results show that the GPHH algorithm is promising. Even though it has not outperformed the benchmarks that are used as reference, this algorithm has better performance than the two basic heuristics, namely Palmer Algorithm and Gupta Algorithm.

Currently, we are working on the application of genetic programming hyper-heuristics for multi-objective FSSP. Not only makespan, but we also consider the lateness (tardiness and earliness) of completing the total jobs.

ACKNOWLEDGEMENTS

This work was supported by Indonesian Ministry of Research, Technology and Higher Education (RistekDikti) under research scheme Penelitian Terapan Unggulan Perguruan Tinggi year 2019-2021 Contract Nr. III/LPPM/2020-04/105-PE-S.

REFERENCES

[1] Cecilia E. Nugraheni and Luciana Abednego. On the Development of Hyper Heuristics Based Framework for Scheduling Problems in Textile Industry. International Journal of Modeling and Optimization, Vol. 6, No. 5, October 2016.

[2] Robert, N. Tomastik, Peter, B. Luh, and Guandong, Liu. Scheduling Flexible Manufacturing System for Apparel Production. IEEE Transaction on Robotics and Automation. 12(5): 789-799.

[3] Scholz-Retter Bernd et al. 2015. Applying Autonomous Control in Apparel Manufacturing. Proc. Of 9th WSEAS Int. Conference on Robotics, Control and Manufacturing Technology. 73-78.

[4] C. E. Nugraheni and L. Abednego, "A survey on heuristics for scheduling problem in textile industry," in Proc. ICEAI 2015.

[5] C. E. Nugraheni and L. Abednego, "A comparison of heuristics for scheduling problems in textile industry," Jurnal Teknologi, vol. 78, no. 6-6. 2016.

[6] Said Aqil and Karam Allali. Three metaheuristics for solving the flow shop problem with permutation and sequence dependent setup time. Proc. Of Conference: 2018 4th International Conference on Optimization and Applications (ICOA). 2019.

[7] Peter Bamidele Shola and Asaju La'aro Bolaji. A metaheuristic for solving flowshop problem. International Journal of Advanced Computer Research, Vol 8(37).

[8] Le Zhang and Jinnan Wu. A PSO-Based Hybrid Metaheuristic for Permutation Flowshop Scheduling Problems. The Scientific World Journal. Vol. 2014.

[9] Ochoa G., Rodriguez J.A.V, Petrovic S., and Burke E. K. 2009. Dispatching Rules for Production Scheduling: a Hyper-heuristic Landscape Analysis. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009), Montreal, Norway.

[10] C. E. Nugraheni and L. Abednego, "Collaboration of multi-agent and hyper-heuristics systems for production scheduling problem,"International Journal of Computer, Electrical, Automation, Control and Information Engineering, vol. 7, no. 8, pp. 1136-1141, 2013.

[11] C. E. Nugraheni and L. Abednego, "A combined meta-heuristic with hyper-heuristic approach to single machine production scheduling," International Journal of Computer, Electrical, Automation, Control and Information Engineering, vol. 8, no. 8, pp. 1322-1326, 2014.

[12] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. European Journal of Operational Research 47 (1990) pp. 65-74.

AUTHORS

Cecilia E. Nugraheni received her bachelor degree (1993) and master degree (1995) from Dept. of Informatic Engineering, Bandung Institute of Technology (ITB), Bandung, Indonesia. She has received PhD Degree (2004) from Dept. of Informatics, Ludwig Maximilians Universität, Munich, Germany. Her research interest includes formal methods, intelligent systems, machine learning, meta-heuristic and hyper-heuristic techniques.

Luciana Abednego received her bachelor degree from Dept. of Informatics, Parahyangan Catholic University, Bandung, Indonesia. She has done her Master in Informatics from Bandung Institut of Technology, Bandung, Indonesia. Currenty, she is working as a lecturer at the Dept. of Informatics, Parahyangan Catholic University. Her research interest includes machine learning and intelligent systems.

Maria Widyarini received her PhD from School of Business Management, Bandung Institute of Technology (ITB). She was involved in SME particularly as trainer and researcher in microfinancing issues. Her specializing is in Microfinance for SME. She was Director of Center of Excellence – Small Medium Enterprise and Development (COE SMED) and Head of BA and MBA Dept in Parahyangan Catholic University (2018 - up to now).