

A COMBINATION OF PALMER ALGORITHM AND GUPTA ALGORITHM FOR SCHEDULING PROBLEM IN APPAREL INDUSTRY

Cecilia E. Nugraheni¹, Luciana Abednego¹ and Maria Widyarini²

¹Dept. of Computer Science, Parahyangan Catholic University, Bandung, Indonesia

²Dept. of Business Adm., Parahyangan Catholic University, Bandung, Indonesia

ABSTRACT

The apparel industry is a class of textile industry. Generally, the production scheduling problem in the apparel industry belongs to Flow Shop Scheduling Problems (FSSP). There are many algorithms/techniques/heuristics for solving FSSP. Two of them are the Palmer Algorithm and the Gupta Algorithm. Hyper-heuristic is a class of heuristics that enables to combine of some heuristics to produce a new heuristic. GPHH is a hyper-heuristic that is based on genetic programming that is proposed to solve FSSP [1]. This paper presents the development of a computer program that implements the GPHH. Some experiments have been conducted for measuring the performance of GPHH. From the experimental results, GPHH has shown a better performance than the Palmer Algorithm and Gupta Algorithm.

KEYWORDS

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

1. INTRODUCTION

The textile industry can be divided into two major segments, namely the textile industry and the apparel industry [2, 3]. The focus of the textile industry is the production of fabric from raw material. It is done through a series of processes such as spinning, weaving, knitting, etc. Meanwhile, the focus of the apparel industry is the transformation of 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 the 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 is to find a sequence of jobs that meets certain optimal criteria. Table 1 illustrates a small FSSP.

There are three jobs J1, J2, and J3 to be processed by five machines M1, M2, M3, M4, and M5. The processing times required for every machine to process each job are given in the table cells.

Table 1. An example of FSSP.

	M1	M2	M3	M4	M5
J1	3	2	2	2	1
J2	2	3	2	2	3
J3	2	1	2	3	5

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 Come First Served - FCFS, Last Come First Served - LCFS, Shortest Processing Time - SPT, Longest Processing Time - LPT, etc.). The application of FCFS and LCFS on the FSSP in Table 1 results in a schedule with a makespan of 21 time units and 17 time units, respectively. The corresponding Gantt Chart of each schedule are shown in Figure 1 and Figure 2. For this case, the application of SPT will give the same result as FCFS and the application of LPT will give the same result as LCFS. There are also several algorithms that can be used for solving FSSP. Some of them are NEH, Palmer, Gupta, Dannenbring, Pour, etc.[4,5,6].

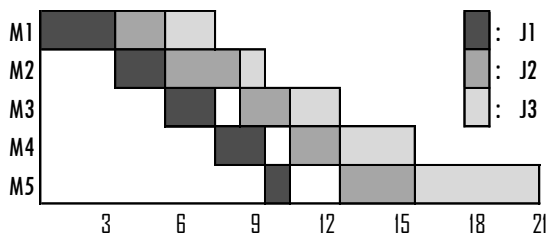


Figure 1. Gantt chart for the solution resulted by using FCFS or SPT.

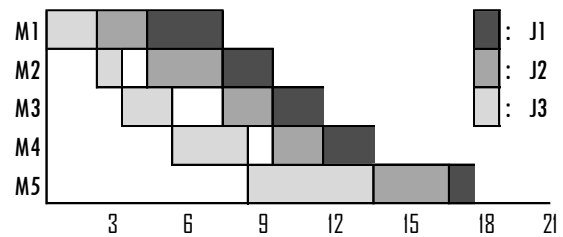


Figure 2. Gantt chart for the solution resulted by using LCFS or LPT.

The second heuristic 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]. Hyper-heuristic is different from the other heuristics. It does not work directly on the problem domains, rather on the heuristics. This is the reason why it 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. Continuing this 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. Some materials of this paper have been published in [12]. Also, this paper contains also the experimental result for testing the correlation of two parameters of the GPHH algorithm that has not been reported.

The rest of this paper is organized as follows. Section 2 describes the genetic programming based hyper-heuristic for FSSP. First, we briefly explain the Palmer Algorithm and the Gupta. Then the principle of genetic programming is given. Next is the explanation of the GPHH algorithm

followed by its implementation. A more complete exposition of this material can be found in [1, 12]. Section 3 discusses the experimental results. Last, conclusion and future work are given in Section 4.

2. GENETIC PROGRAMMING BASED HYPER-HEURISTIC FOR FSSP

2.1. Palmer Algorithm and Gupta Algorithm

Palmer Algorithm and Gupta Algorithm are two heuristics for solving FSSP. The principle of both algorithms is similar. Each of them consists of two steps: computing the slope index of each job and ordering the jobs based on the slope indices. The difference between these algorithms is the formula for counting the slope index. Let A_j denote the slope index of job j and p_{ij} denote the processing time required by machine j for processing job i , the formula for counting the slope indices by Palmer Algorithm and Gupta Algorithm is given in Eq. 1 and Eq. 2, respectively.

$$A_j = - \sum_{i=1}^m \{m - (2i - 1)\} p_{ij} \quad \text{Eq. 1}$$

$$A_j = \frac{e_j}{\min_{1 \leq k \leq m-1} \{p_{jk} + p_{j(k+1)}\}} \quad \text{Eq. 2}$$

where $e_j = 1$ whenever $p_{j1} < p_{jm}$ and $e_j = -1$ whenever $p_{j1} \geq p_{jm}$.

Using Palmer Algorithm and Gupta Algorithm for solving FSSP given in Table 1 will give the same result as FCFS and as LCFS, respectively.

2.2. Genetic Programming

One of the search algorithms that belong to meta-heuristic is the genetic algorithm. This algorithm is inspired by Charles Darwin's theory of natural evolution and reflects the process of natural selection. A population of individuals representing solutions to an optimization problem is evolved toward better solutions. Each individual has a set of properties called chromosome that can be mutated and altered. The fittest individuals are selected for reproduction to produce offspring of the next generation.

Genetic programming uses the same idea as the genetic algorithm. It is inspired by Darwin's theory of evolution. Like genetic algorithm, genetic programming uses chromosomes for representing the candidate solutions. A chromosome in genetic programming is a computer program (function) that can be used to find solutions to problems. Moreover, genetic programming also uses genetic operators to make modifications or alterations to the individuals. The application of genetic operators (crossover, mutation, and reproduction) generates new computer programs.

Figure 3 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 the original tree based on a mutation point. Figure 4 shows an example of a mutation operation over a computer program. The constant 2 is replaced by the function $p+g$. This generates a new computer program which is $(p/1 - (p + g))$.

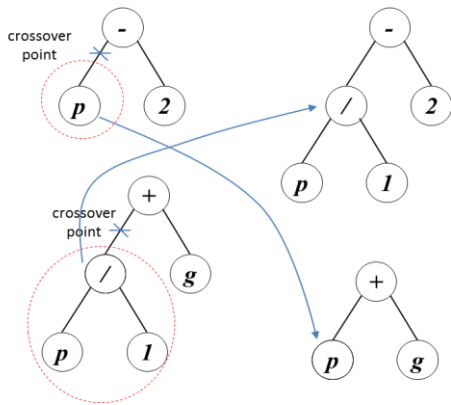


Figure 3. An example of crossover operation.

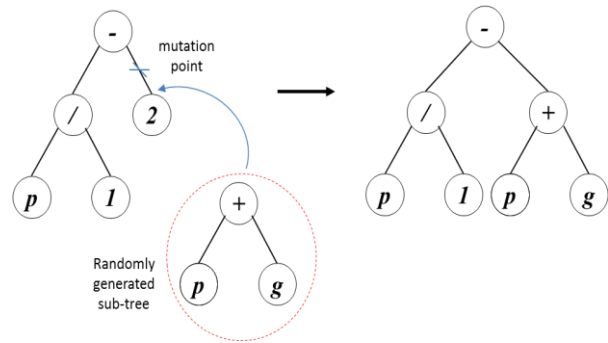


Figure 4. An example of mutation operation.

2.3. Genetic Programming Hyper-heuristic (GPHH)

The algorithm of GPHH is given in Figure 5. 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.

function GPHH (*ops*, *oprs*, *popSize*, *poolSize*, *depth*, *maxRun*, *maxGen*, *problem*, *COrate*, *Mrate*) \rightarrow (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 5. GPHH Algorithm.

2.4. 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 program is given in Figure 6. 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 7.

Job	Machi...	Machi...	Machi...	Machi...	Machi...
1	54	79	16	66	58
2	83	3	89	58	56
3	15	11	49	31	20
4	71	99	15	68	85
5	77	56	89	78	53
6	36	70	45	91	35
7	53	99	60	13	53
8	38	60	23	59	41
9	27	5	57	49	69
10	87	56	64	85	13
11	76	3	7	85	86
12	91	61	1	9	72
13	14	73	63	39	8
14	29	75	41	41	49
15	12	47	63	56	47
16	77	14	47	40	87
17	32	21	26	54	58
18	87	86	75	77	18
19	68	5	77	51	68
20	94	77	40	31	28

Figure 6. The problem tab for displaying the problem.

Gupta :
 - Makespan = 1400.0
 - Schedule: 10 - 2 - 8 - 16 - 14 - 15 - 13 - 0 - 7 - 3 - 4 - 5 - 9 - 17 - 1 - 18 - 6 - 19 - 12 - 11

Palmer :
 - Makespan = 1384.0
 - Schedule: 8 - 10 - 16 - 14 - 15 - 18 - 2 - 5 - 13 - 7 - 1 - 3 - 0 - 4 - 12 - 6 - 11 - 9 - 17 - 19

GPHH :
 - Makespan = 1384.0
 - Schedule: 8 - 10 - 16 - 14 - 18 - 15 - 2 - 5 - 13 - 7 - 1 - 3 - 0 - 4 - 12 - 6 - 11 - 9 - 17 - 19

Figure 7. The solution tab for displaying the schedule and makespan of each algorithm.

3. EXPERIMENTAL RESULTS AND ANALYSIS

In order to measure the performance of the heuristics generated by GPHH, a number of experiments were carried out. As benchmark, we have taken the proposed by Taillard et al. [13]. This benchmark provides a number of scheduling problems grouped by the number of jobs and machines as shown in Table 2. 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

Table 2. 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 8, Figure 9, and Figure 10 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 (Figures 8, 9, and 10), some values resulted from GPHH are worse than the one resulted from the Palmer Algorithm and Gupta Algorithm. Whether cases with many very large jobs will reduce the performance of GPHH still needs to be further analyzed and tested.

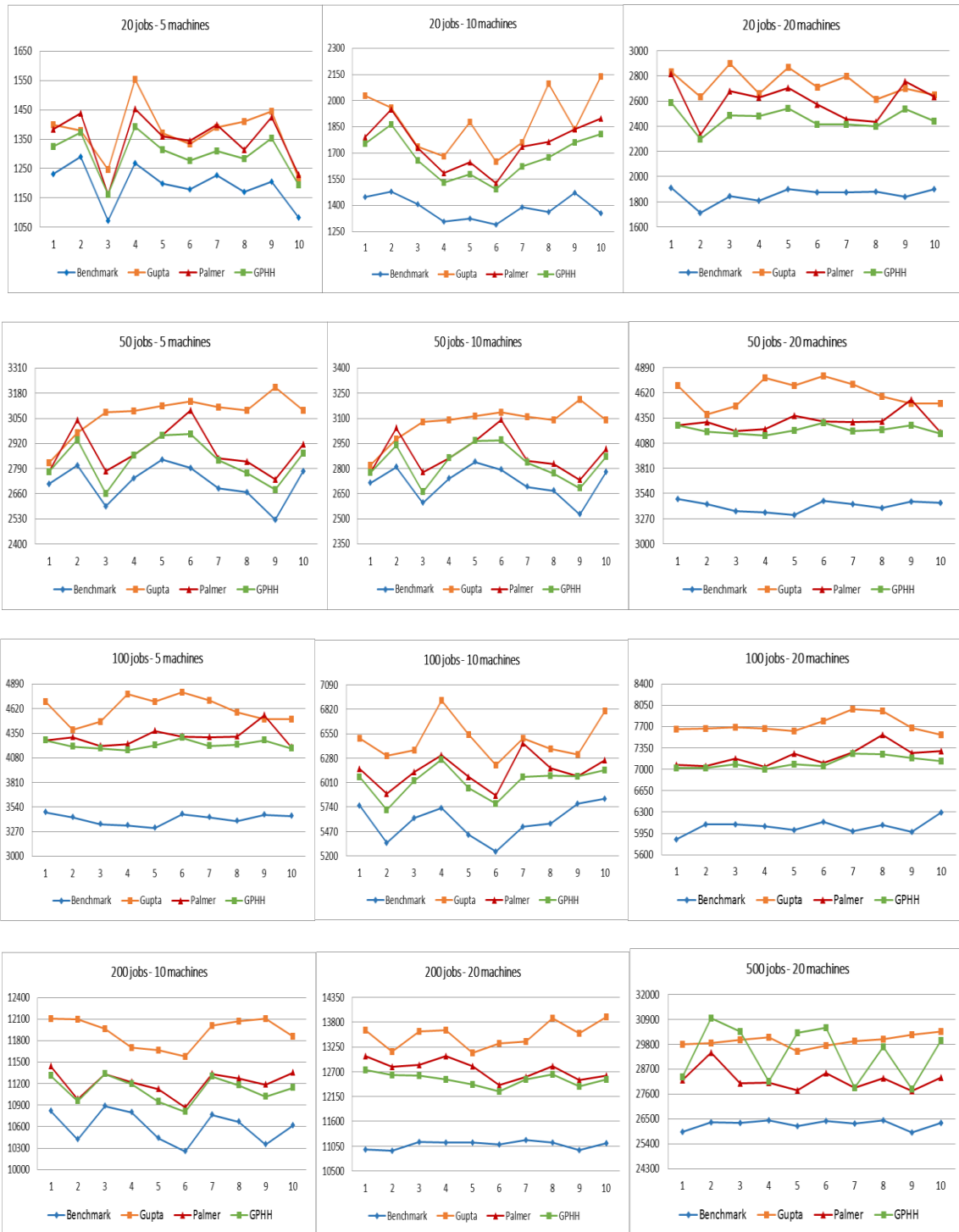


Figure 8. Experimental Results for Cross Over Rate 0.75.

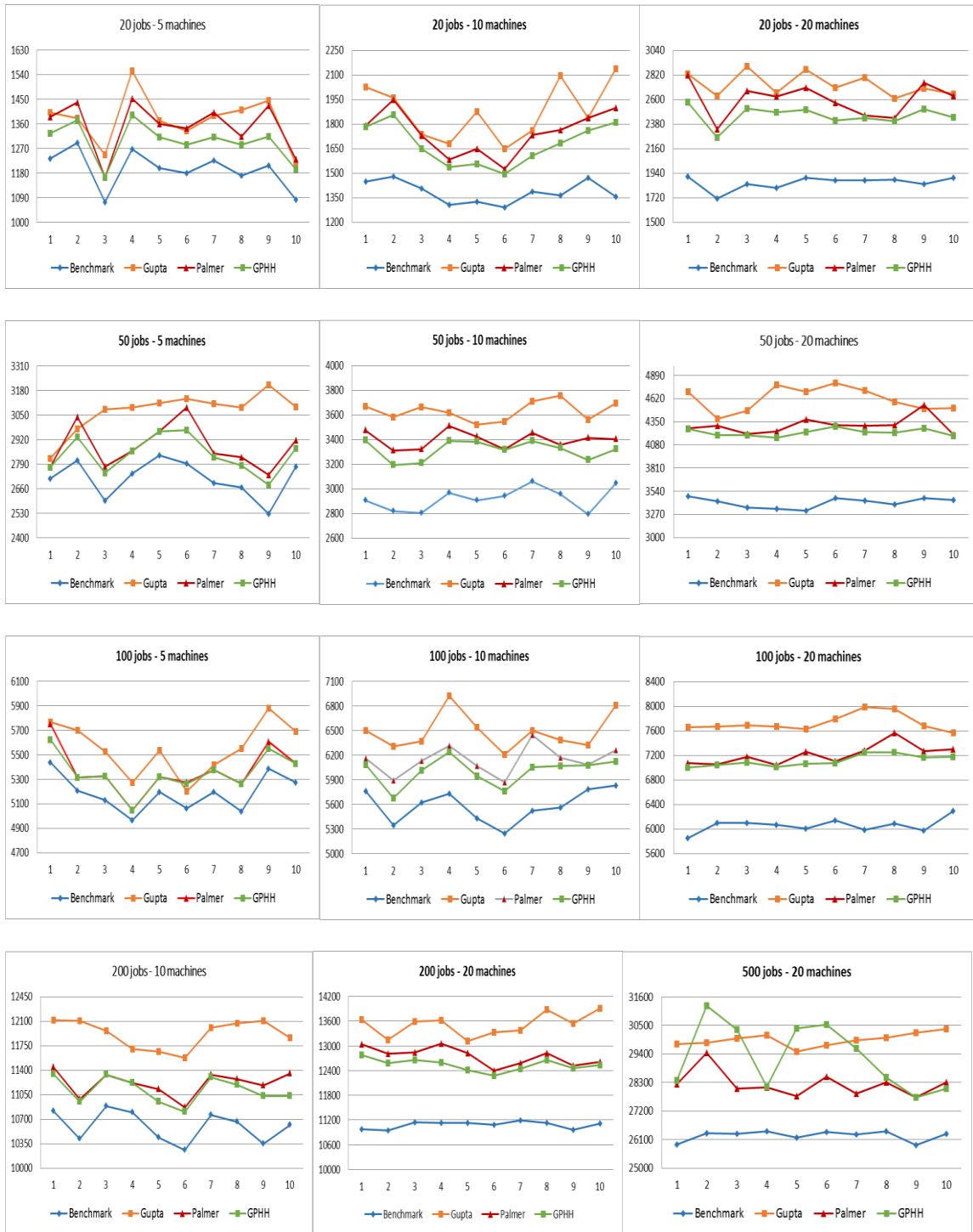


Figure 9. Experimental Results for Cross Over Rate 0.80.

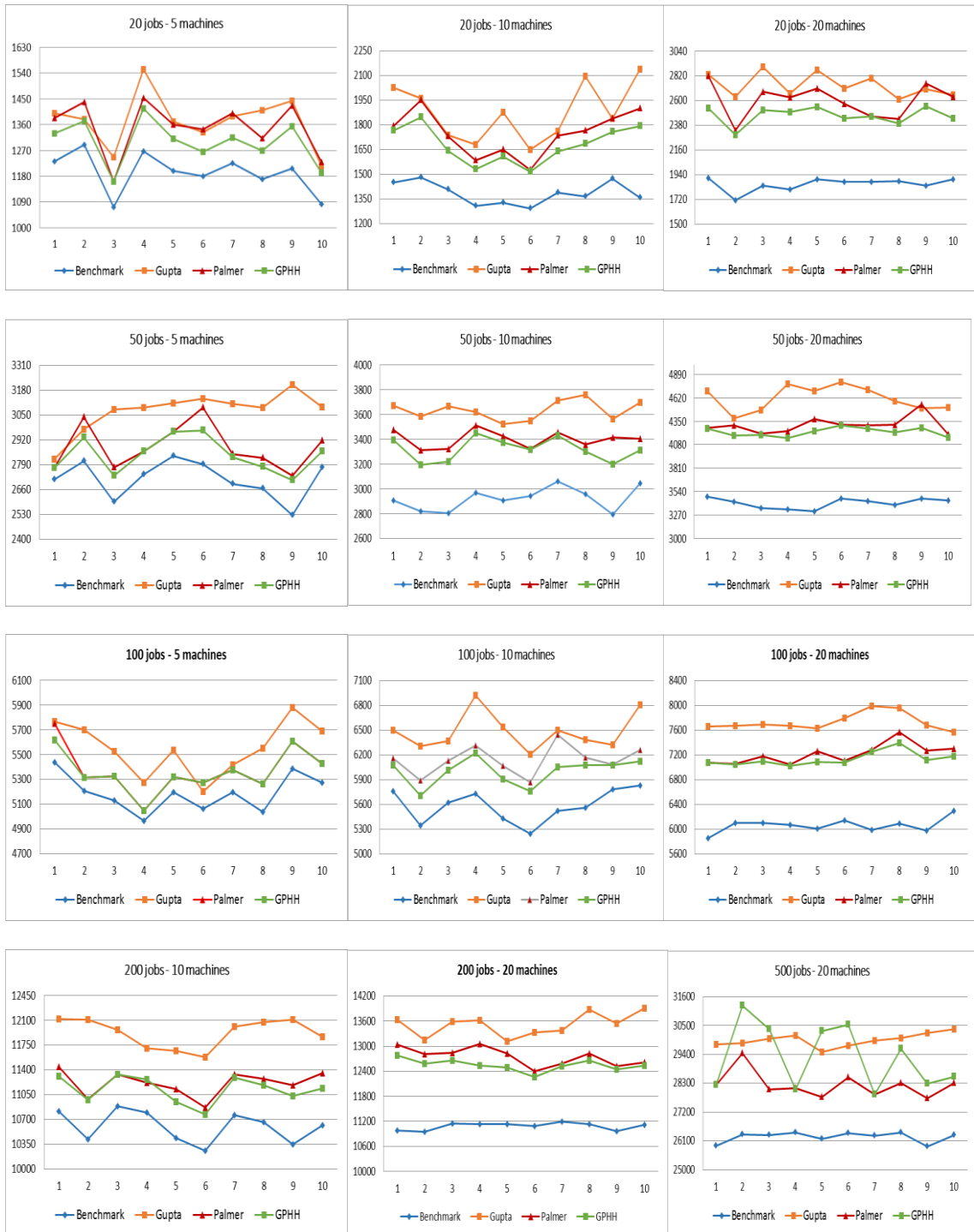


Figure 10. Experimental Results for Cross Over Rate 0.85.

In terms of the depth of the syntax tree, from Figure 11 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 12.

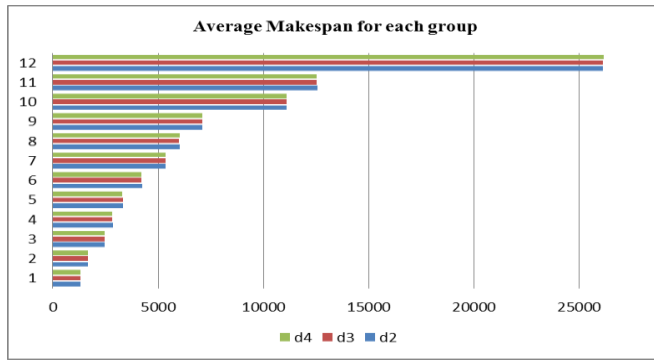


Figure 11. Average Makespans for each problem instance group.

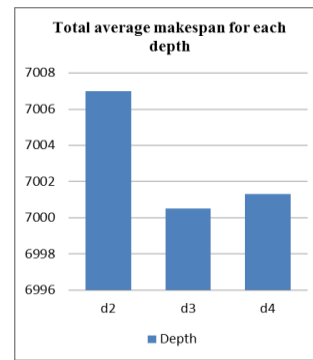


Figure 12. Total average for each syntax tree depth.

For further analysis, especially for large cases which is 500 jobs and 20 machines, we have conducted another experiment. This experiment aims to find the correlation between the size of the pool and the resulted makespan. The hypothesis is the larger the pool-size, the more variants of individual computer programs and moreover, the better heuristics will be generated by the algorithm. For each problem instance with 500 jobs and 20 machines, we run the program with three different pool size: 250, 300, and 400. The result is given in Figure 13. It shows that the hypothesis holds. In average, the pool size of 400 yields the best performance.

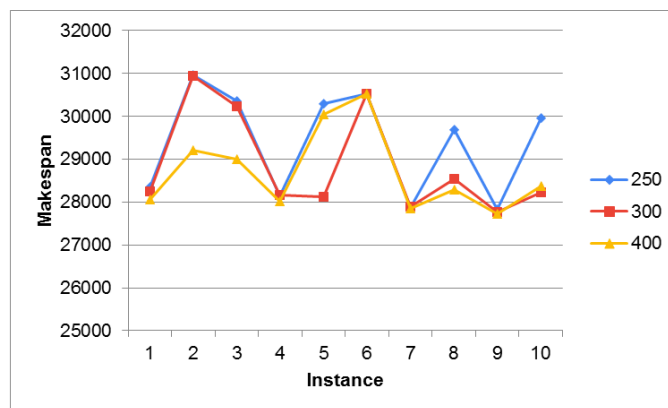


Figure 13. Total average for each syntax tree depth.

4. 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] C.E. Nugraheni, L. Abednego, and M. Widyarini. A Genetic Programming based Hyper-Heuristic for Production Scheduling in Apparel Industry. *International Conference on Machine Learning Techniques and NLP (MLNLP 2020)*, October 24-25, 2020, Sydney, Australia Volume Editors : David C. Wyld, Dhinakaran Nagamalai (Eds) ISBN : 978-1-925953-26-8.
- [13] 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).

