On the Development of Hyper Heuristics Based Framework for Scheduling Problems in Textile Industry

Cecilia E. Nugraheni and Luciana Abednego

***Abstract*—Textile industry, which is one of the most prominent industries in Indonesia, faces a problem caused by the condition of machine productions. This situation leads to a need of good machine scheduling system. Generally, production processes in textile industry belong to the flow shop scheduling problems (FSSP). Many approaches/heuristics have been proposed for solving FSSP. Two of them are Palmer's algorithm and Gupta's algorithm. This paper investigates a method, called genetic algorithm hyper-heuristic, for combining those heuristics in order to obtain some new better heuristics. This method is then implemented in a framework.**

***Index Terms*—Scheduling, flowshop scheduling problem (FSSP), heuristics, hyper-heuristics, genetic programming.**

I. Introduction

Textile industry belongs to the most prominent industries in Indonesia. One important problem that should be tackled by this industry is the condition of production machines. The relatively old machines (around 20 years old) can cause a large amount of energy consumption and affect the optimality of working speed and quality of the products. Good production scheduling can be a solution for this problem.

Like in many other manufacture industries, in textile industry, production scheduling plays an important role. In this context, scheduling is understood as assigning jobs to machines or human (such as operators) for specified time period satisfying some constraints.

The scheduling problem in textile industry generally belongs to the flow shop scheduling problem (FSSP). Given m machines and n jobs that will be processed on each machine, an FSSP is the problem to find a sequence of jobs that satisfies some particular criteria. One of the important objectives is to find the minimum makespan. Makespan is the time between the beginning of the execution of the first job of the sequence on the first machine and the completion of the execution of the last job of the sequence on the last machine.

There are many proposed algorithms/approaches/methods for solving FSSP that can be found in the literature. These methods can be classified into two techniques: constructive or improvement [1], [2]. The difference between the two techniques lies on how the job sequence is constructed. A constructive technique iteratively builds the sequence by adding jobs one by one into the sequence. Some examples of these techniques are FCFS, Johnson's algorithm, Gupta's algorithm, Palmer's algorithm, NEH, CDS, Dannenbring's algorithm, Pour's algorithm, and MOD [3]-[8]. These algorithms are usually called (low level) heuristics. An improvement technique starts with a solution candidate (an arbitrary job sequence) and then by using particular mechanism it iteratively changes the sequence in order to find a best solution. Some examples of these techniques are genetic algorithm, simulated annealing, tabu search, etc. Improvement techniques are sometimes also called metaheuristics.

Each heuristic has strengths and weaknesses. There are some approaches for combining heuristics in order to obtain some new better heuristics. One of them is hyper-heuristic. A hyper-heuristic is a heuristic search method that seeks to automate the process of selecting, combining, generating or adapting several simpler heuristics to efficiently solve computational search problems. The important feature of hyper-heuristics is that they search a space of heuristics rather than a space of solutions directly.

We are interested in developing a hyper-heuristics based framework that can be used to solve FSSPs. For achieving the goal, some research have been conducted. At the first stage, a survey on some low-level heuristics for FSSP was done [8]. Nine heuristics have been collected and applied to some simple scheduling problems related to textile industry. At the second stage, a prototype program implementing all the heuristics has been developed. Using the program, some experiments to compare the performance of the heuristics have been conducted [9]. At the third stage, the current stage, we are developing a framework for solving FSSP using the principle of hyper-heuristic. In particular, we use genetic programming hyper heuristic in this work. We try to combine two low level heuristics that share the similar principle, which are Palmer's algorithm and Gupta's algorithm. It is expected that this framework can produce some new heuristics that are better than the original Palmer's algorithm and Gupta's algorithm.

There are many work dedicated to the solution of FSSP found in the literature. The contribution of our work is the using of genetic programming hyper heuristics techniques to combine Palmer's algorithm and Gupta's algorithm. This same topic, as far as our knowledge, is not yet reported in the literature.

The rest of this paper is structured as follows. Section II gives a brief explanation of Palmer's algorithm and Gupta's algorithm based on [5], [9]. Section 3 describes the basic terminology and principle of genetic programming according to [10]. Section IV presents the framework design. Section 5Vconcludes the paper.

Manuscript received May 25, 2016; revised September 18, 2016.

The authors are with the Informatics Dept., Parahyangan Catholic University, Bandung, Indonesia (e-mail: {cheni, Luciana}@unpar.ac.id).

DOI: 10.7763/IJMO.2016.V6.539

II. Palmer's Algorithm and Gupta's Algorithm

Given m machines and n jobs that will be processed on each machine, a FSSP objective is to find a sequence of jobs that meets a particular optimality criterion. In this work, makespan is chosen as the objective of the optimality.

There are many approaches to solve FSSP. Among of them are Palmer's algorithm and Gupta's algorithm. Both algorithms use the similar principle in producing the job sequence. The ordering is based on a value or slope index. Each algorithm consists of two steps, as follow:

1) Step 1: For n job and m machine FSSP, compute slope index A_j for jth job by using a particular rule.
2) Step 2: Order the jobs in the sequence based on descending (decreasing) order of Aj values.

Let p_{ij} denote the processing time required by machine j for processing job i, for step 1, Palmer's algorithm uses this following rule:

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

whereas Gupta's algorithm uses this following rule

$$A_j = \frac{e_j}{\min\limits_{1 \le k \le m-1}\left\{p_{jk}+p_{j(k+1)}\right\}} \quad \text{(Eq. 2)}$$

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

To give an illustration on how the Palmer's algorithm and Gupta's algorithm work, let's take a small FSSP. There are three jobs that will be processed on five machines. The processing times required for every job by every machine are given in Table I.

TABLE I: Processing Times

Job	Machine				
	m1	m2	m3	m4	m5
j1	6	5	3	9	5
j2	8	1	8	5	6
j3	2	1	3	8	6

The slope index of every job can be computed using Eq.1 for Palmer's algorithm and Eq. 2 for Gupta's algorithm. The computation results are given in Table II.

TABLE II: Slope Indices

Job	Algorithm	
	Palmer	Gupta
j1	4	-1/8
j2	0	-1/9
j3	30	1/3

The job ordering is then can be done based on the slope index values in Table II. The ordering resulted by Palmer's algorithm is j3-j1-j2 with makespan 36 and by Gupta's algorithm is j3-j2-j1 with makespan 35. The corresponding Gantt chart of each resulted schedule is given in Fig. 1 and Fig. 2. In this case, Gupta's algorithm produces a better solution since the makespan of its resulted scheduling is smaller.

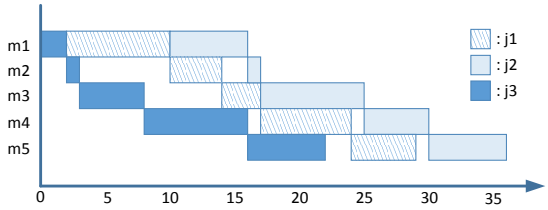

Fig. 1. Gannt chart of the schedule resulted by Palmer's algorithm.

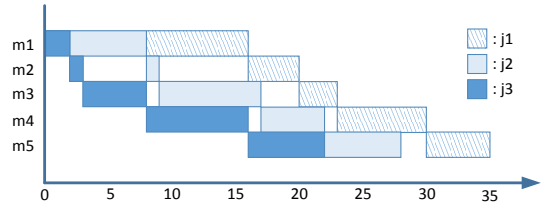

Fig. 2. Gannt chart of the schedule resulted by Gupta's algorithm.

III. Genetic Programming

Genetic programming is an extension of genetic algorithm. They are inspired by biological evolution, in particular by Darwin's theory about evolution. In solving a problem, individual in genetic algorithm represents a candidate solution. This individual is treated as a chromosome. Starting with an initial population consisting a number of individual, the algorithm iteratively produces a new population by applying several genetic operations (reproduction, crossover, and mutation). The individual with the best fitness is chosen to be the solution.

Genetic programming shares the similar idea. It differs in the role of the chromosomes. Instead of representing the candidate solution for a problem domain, an individual or the chromosome represents a computer program that later can be used to solve the problem.

In genetic programming, a computer program is usually represented as a syntax tree. The leaves, or terminals, represent operands and the branch nodes represent operators or functions. For example, a computer program *max* (1/(*x*+*y*), 2**y*) can be represented as a tree shown in Fig. 3.

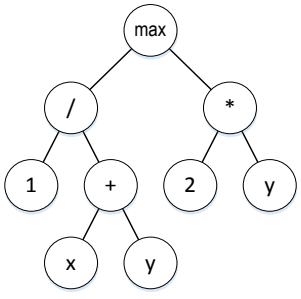

Fig. 3. Syntax tree for *max*(1/(*x*+*y*), 2**y*).

How genetic programming works is shown in Fig. 4. Very similar to genetic algorithm, it starts with an initial random population consisting of a number of individual representing a set of computer programs. Then, iteratively it produces new population. The new individual, which is a computer

program, is resulted from applying some genetic operations: reproduction, crossover, mutation, or architecture altering.

Fig. 5 illustrates how genetic programming produces a new program by applying crossover operation. First, cross over points are defined for each parent and then the subtree from each parent tree is exchanged.

How a mutation operation works is illustrated in Fig. 6. A mutation point is defined in the beginning and a new subtree is then randomly generated. The subtree from the initial program is then replaced by the generated subtree.

There are five preparatory steps of genetic programming, which are:

1) The set of terminals (e.g. the independent variables of the problem, zero-argument functions, and random constants) for each branch of the to-be-evolved program.
2) The set of primitive functions for each branch of the to-be-evolved program.
3) The fitness measure for measuring the fitness of individuals in the population,
4) Certain parameters for controlling the run, and
5) The termination criterion and method for designating the result of the run.

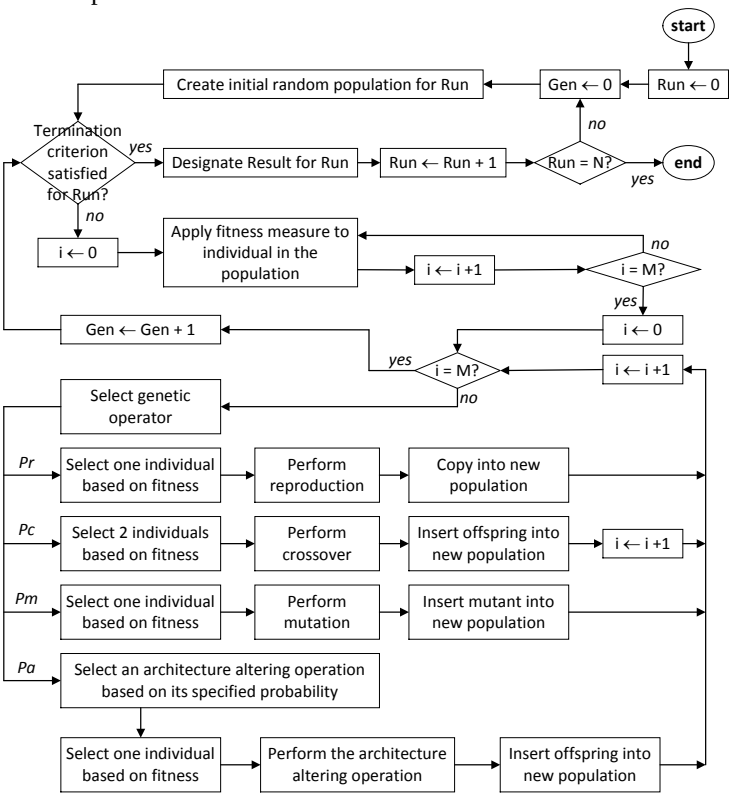

Fig. 4. Flowchart of genetic programming.

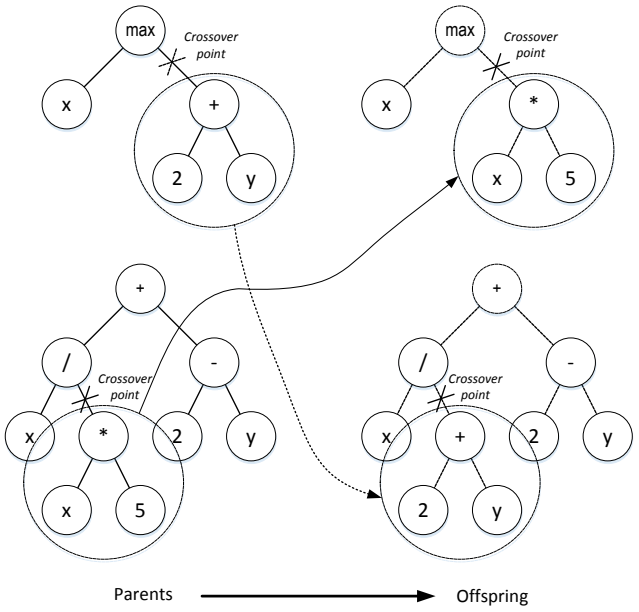

Fig. 5. Example of two new individual crossover between two syntax trees.

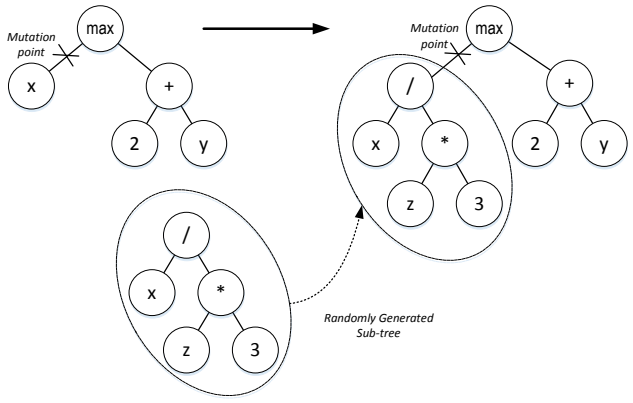

Fig. 6. Example of sub-tree mutation.

IV. Framework Design

Inspired by our previous work in [11] and [12], using the multi-agent system in developing a system, a framework have been designed as shown in Fig. 7. The framework is simpler than in [11] and [12] since only one algorithm agent

is implemented. The framework is composed of five agents, namely: problem agent, trainer agent, training dataset agent, algorithm agent – GPHH, and solver agent.

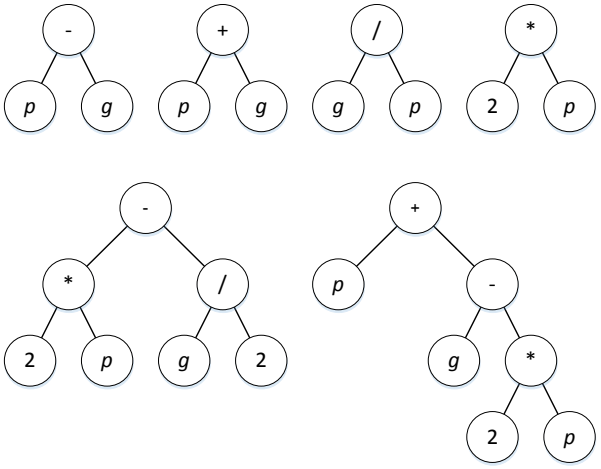

Fig. 6. The example rules represented as syntax trees.

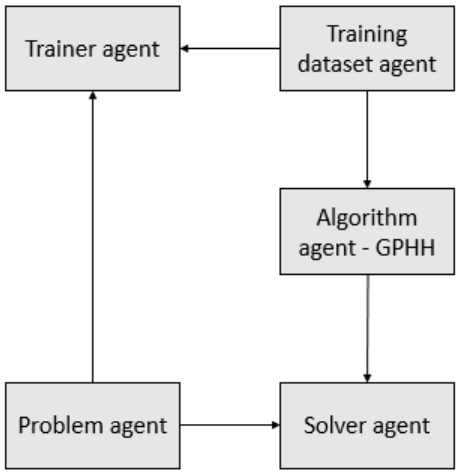

Fig. 7. The architecture of the genetic programming hyper-heuristics for FSSP framework.

Problem agent is the entry point of the system. It is responsible for initializing all other agents by sending the problem description to the trainer agent. Trainer agent trains the system with a group of training dataset based on the problem description got from the problem agent. All training datasets are managed by training dataset agent. This agent also provides training dataset to algorithm agent, in this case, GPHH (genetic programming hyper-heuristic). Algorithm agent is responsible for running the genetic programming and sending the best solution found to solver agent. The solver agent solves the problem from the problem agent by using the best heuristic got from the algorithm agent.

As mentioned before, the framework is expected to produce some heuristics using Palmer's algorithm and Gupta's algorithm as reference. The job ordering of Palmer's algorithm and Gupta's algorithm is based on a value called slope index. Let p and g denote the slope index of Palmer's algorithm and Gupta's algorithm, respectively, the basic idea is simply to make the framework produce rules containing p and/or g. Some examples of the rules are p-g, p+g, g/p, 2*p, 2*p-g/2, p+(g-2*p). Fig. 8 shows the corresponding syntax trees of those example rules.

Some parameter settings are already defined at preparatory stage of genetic programming as required. The terminal set consists of two constants (1 and 2) and two variable symbols (p and g). The set of primitive functions consists of four arithmetic operator symbols: +, -, *, and /. The fitness of each individual is resulted by executing its corresponding program on a set of data training. However, we have not yet defined the number of the run and the generation. Both parameters will be set later during experiment stage.

V. Conclusion

Scheduling problems in textile industry in general belong to the Flowshop Scheduling Problem (FSSP). Given m machines and n jobs, the objective of FSSP is to find a job ordering that meets some particular criteria.

In this work, a framework that can be used to solve FSSP has been designed. This framework implements the genetic programming hyper-heuristic techniques for producing new heuristics and is developed using multi-agent system approach. The architecture, some parameter setting, and the work principle of the framework are reported in this paper.

The current stage is the implementation stage. The framework is being developed in Java programming language. Next step, some experiments will be conducted in order to get the best rules. In the future, it is planned to consider more heuristics for FSSP such as NEH, CDS, Pour's algorithm, and other heuristics.

References

[1] V. Modrák and R. S. Pandian, "Flow shop scheduling algorithm to minimize completion time for n-Jobs m-Machines problem," *Technical Gazette*, vol. 17, no. 3, pp. 273-278.

[2] Rubén Ruiz and Concepción Maroto, "A comprehensive review and evaluation of permutation flowshops heuristics," *European Journal of Operational Research*, vol. 165, pp. 479-494, 2005.

[3] Pavol Semanco and Vladimir Modrak, "A comparison of constructive heuristics with the objective of minimizing makespan in the flow-shop scheduling problem," *Acta Polytechnica Hungarica*, vol. 9, no. 5, 2012.

[4] E. Taillard, "Some efficient heuristic methods for the flow shop sequencing problem," *European Journal of Operational Research*, vol. 47, pp. 65-74, 1990.

[5] Thomas Morton, *Heuristic Scheduling Systems: With Application to Production Systems and Project Management*, John Wiley & Sons, 1993, vol. 3.

[6] F. Wang, Y. Rao, and Y. Hou, "Design and application of a new hybrid heuristic algorithm for flow shop scheduling," *I. J. Computer Network and Information Security*, vol. 2, pp. 41-49, 2011.

[7] A. Sulaksmi, Annisa Kesy Garside, "dan Fithriany Hadziqah. Penjadwalan Produksi dengan Algoritma Heuristik Pour (Studi Kasus: Konveksi One Way Malang)," *Jurnal Teknik Industri*, vol. 15, no. 1, pp. 35-44, Feb. 2014.

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

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

[10] John R. Koza and R. Poli. Genetic Programming. [Online]. Avaiable: http://cswww.essex.ac.uk/staff/rpoli/papers/ KozaPoli2005.pdf.

[11] 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.

[12] 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.

Cecilia E. Nugraheni received the bachelor and master degrees in informatics engineering from Institut Teknologi Bandung, Indonesia. She received the Ph.D. degree in informatics from Institut für Informatik, Ludwig-Maximilians Universität, Munich, Germany. Currently, she is a lecturer at Computer Science Dept. Parahyangan Catholic University, Bandung, Indonesia. Her research interests are formal specification and verification of reactive systems, multi agent systems, production scheduling optimization, meta- and hyper-heuristics.

Luciana Abednego received the bachelor degree in computer science from Parahyangan Catholic University. She received the master degree in informatics engineering from Institut Teknologi Bandung. Currently, she is a lecturer at Computer Science Dept. Parahyangan Catholic University, Bandung, Indonesia. Her research interests include programming and algorithms, computer graphics, production scheduling optimization, meta- and hyper-heuristics.