International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:4, No:12, 2010

A Study of Cooperative Co-evolutionary Genetic Algorithm for Solving Flexible Job Shop Scheduling Problem

Lee Yih Rou, and Hishammuddin Asmuni

***Abstract*—**Flexible Job Shop Problem (FJSP) is an extension of classical Job Shop Problem (JSP). The FJSP extends the routing flexibility of the JSP, i.e assigning machine to an operation. Thus it makes it more difficult than the JSP. In this study, Cooperative Co-evolutionary Genetic Algorithm (CCGA) is presented to solve the FJSP. Makespan (time needed to complete all jobs) is used as the performance evaluation for CCGA. In order to test performance and efficiency of our CCGA the benchmark problems are solved. Computational result shows that the proposed CCGA is comparable with other approaches.

***Keywords*—**Co-evolution, Genetic Algorithm (GA), Flexible Job Shop Problem(FJSP)

I. Introduction

TODAY's manufacturing industries are concerned not only on the cost and quality of the product, but they are also concern about the delivery performance of the product. Besides that, the delivery performance also becomes a tool to secure competitive advantages. Therefore scheduling plays an important role in the manufacturing process. A schedule is an allocation of the operation to the time intervals on the machines. To find a best schedule it can be either very easy or very difficult, and it depends on the process constraint, shop environment and performance indicator (makespan, machine workload). Job Shop Problem (JSP) is a branch of production manufacturing and it is a hardest combinatorial problem. The classical JSP consist of *n* jobs and *m* machines and each job has a sequence of operations. The problem of the JSP is the sequence of the operations on the machine in order to find a minimum makespan (time needed to complete all jobs).

In order to make the JSP closer to the real world of the manufacturing system, the JSP is extended to Flexible Job Shop Problem (FJSP). In FJSP an operation can be processed by more than one machines, but in the JSP one operation can be processed by exactly one machine. Thus the FJSP present two difficulties:

i. Machine selection problem, assigned a suitable or appropriate machine to an operation.
ii. Operation sequencing problem, sequence the operation on the machine in order to find a minimum makespan.

Lee Yih Rou. Author is with the Software Engineering Department, Universiti Teknologi Malaysia, Skudai 81300 Malaysia(corresponding author to provide phone: 012-427-3568; e-mail: yihrou@gmail.com).

Hishammudin Asmuni Author is with the Software Engineering Department, Universiti Teknologi Malaysia, Skudai 81300 Malaysia (e-mail: hishamudin@utm.my).

Brucker and Schlie[3] were the first to develop the polynomial algorithm for solving the FJSP problem with two jobs. In the recent years, there are a growing number of literatures in the FJSP. The related publication is Chen et al.[5], Dauzère-Pérès and Paulli[6], Kacem et al.[7], Xing et al.[8], and Yadazni et al.[9] [10] among others. Among the literatures, it can be categorized into the hierarchical approach or the integrated approach. The hierarchical approach solves the machine selection problem and operation sequencing problem hierarchical (assign then sequence) hence it reduces the difficulties of the FJSP. Brandimarte[11], solved this FJSP hierarchically. He adopted the dispatching rules to solve the machine selection problem then solved the sequencing problem using the different Tabu Search (TS). However the integrated approaches solve the machine selection problem and operation sequencing problem simultaneously. Dauzère-Pérès and Paulli[6], Hurink et al.[12], Mastrolilli and Gambardella[13] adopted the integrated approach and proposed a different TS for solving the FJSP. In their approach, there is no distinction in solving the problem of machine selection and operation sequence problem.

In recent years the GA has been successfully adopted to solve the FJSP, and this can be proved by the growing number of publication. The relevant works are Mesghouni et al.[14], Chen et al.[5] and Kacem et al.[7]. Mesghouni et al.[14] were the first to model the GA for the FJSP; they proposed the parallel job representation and parallel machines representation. Chen et al.[5] also proposed a new chromosome representation that consists of two strings i.e *A String* and *B String*. *A String* is defined by the routing problem whereas *B String* defines the sequence on the operation problem. Lastly Kacem et al.[7], proposed a task sequencing list as the chromosome representation that combines both the routing and sequencing information. Besides that, they developed an approach by the localization to find a promising initial assignment.

In this study we proposed a cooperative co-evolutionary genetic algorithm (CCGA) for the FJSP. In CCGA, the FJSP is decomposed into two problems (sub problem). Each problem is evolved by a single GA. In this way, two parallel searches on two sub problem are more efficient than a single search on entire problem.

II. Problem Description

FJSP consist of a set of *n* jobs $J = \{J_1, J_2,\ldots,J_n\}$ and processed by m set of machines $M = \{M_1, M_2,\ldots,M_m\}$. A job J_i

is formed by a sequence of operations ($O_{i1,}$ O_{i2},... O_{in}), . Each operation O_{in}, i.e the operation *n* from job *i* can be executed on any machine from the predetermined alternative machine set M_{ij} M. The processing time for each operation O_{in} is predetermined. All jobs and machines are available at time 0. There are three constraints for jobs and machines:

i. There are precedence constraints among the operation of the same job.
ii. Each operation must be completed without interruption once started.
iii. Each machine can only execute one operation at a time.

There are two problems presented in the FJSP that are the machine assignment problem and operation sequencing problem. In the machine assignment problem an appropriate machine is selected and assigned to an operation whereas the operation sequencing problem is to sequence the operation on the machine in order to minimized the makespan, i.e., the time needed to complete all the jobs. Makespan is defined as C_M = max $\{C_i\}$ where C_i is the completion time for job J_i.

The flexibility of the FJSP can be categorized into partial flexibility and total flexibility[7]. In the case of partial flexibility each operation can only be executed by a limited number of machines $M_{ij} \neq M$. However in the case of total flexibility each operation can be processed by any available machines $M_{ij} \subset M$.

Problem instance of the FJSP with partial flexibility is given in TABLE I. In TABLE I, each rows correspond to operations and columns representing the machine. Each unit value in the table is the processing time of the machines. However the symbol "-" means that the machine cannot execute the corresponding operation.

TABLE I PROCESSING TIME TABLE

Job	Operation	Machines				
		M_1	M_1	M_1	M_1	M_1
J_1	O_{11}	2	6	5	4	3
	O_{12}	-	8	-	4	-
J_2	O_{21}	2	2	-	8	-
	O_{22}	8	7	5	4	8
J_3	O_{31}	6	-	-	9	3
	O_{32}	1	-	4	4	-
	O_{33}	7	5	-	6	-
J_4	O_{41}	3	-	6	-	5
	O_{42}	4	6	5	-	-
	O_{43}	8	7	11	5	8

III. COOPERATIVE CO-EVOLUTIONARY GENETIC ALGORITHM FOR FJSP

A. Cooperative co-evolutionary genetic algorithm

Co-evolutionary algorithm introduces the concept of ecosystem that involves two or more interacting species. During the evolution process there are interactions between individual from different species. However, in conventional genetic algorithm (GA) the individual does not interact with the individual from other species. Co-evolutionary algorithm is reported that it provides a promising alternative to a standard evolutionary algorithm in a complex and dynamic problem[15]. Co-evolutionary is categorized into the cooperative co-evolutionary and competitive co-evolutionary. Here, we will focus on the cooperative co-evolutionary algorithm, and it is because of the trend of current research which focuses on the cooperative co-evolutionary algorithm.

In this research we present Cooperative Co-evolutionary Genetic Algorithm (CCGA) for solving the FJSP. The CCGA is first proposed by De Jong and Potter[16] to improve the traditional GA that has a slow evolution process for large search space[17]. CCGA uses the strategy of divide and conquer. CCGA divides or splits a big problem into smaller problems i.e species, each of this species is representing the partial solution of the problem. Each species is maintained in a population that contains different individuals (chromosomes). Furthermore these species evolve independently by a single genetic algorithm. Therefore during the individual's fitness evaluation process, the individual is cooperated with its cooperative partner to form a complete solution to calculate the individual's fitness.

In our algorithm there are two populations, and each population represents the difficulties of the FJSP as mentioned in section II. The first population is the machine selection population *PopM*(*N*)={1,2,…*N*}, and the second population is the operation sequencing population *PopO*(*N*)={1,2,…N}. Since these two populations have different features and therefore the genetic operation and individual representation are different for both populations. Moreover the details of these two populations are explained in sections C and D.

B. Cooperative partner selection and fitness evaluation

There is a great difference between GA and CCGA in the fitness evaluation. In GA, fitness value of an individual is dependent on the quality of the solution and it is evaluated independently. Note that the quality we considered here is the makespan. But in CCGA the individual's fitness depends on how well it cooperates with its cooperative partner. Thus to evaluate the CCGA individual's fitness value, the method to select cooperative partner should be determined first. There are various methods to select the cooperative partner and we have conducted some testing on the randomly select and select the best cooperative partner. The computational result indicates that the randomly select cooperative give a better result among others.

Roulette wheel selection is chosen to select the individual for reproduction. By using this method, individual with higher fitness will have a higher probability to be selected. Thus it has increased the chance to produce individuals with better fitness. The individual's fitness is calculated by using (1), and the fitness value is in the range of 0 to 1.

$$f_q(s) = \frac{g_q(s) - \{max_{u \in PopM[q]} g_q(u)+1\}}{min_{u \in PopM[q]}\ g_q(u) - \{max_{u \in PopM[q]} g_q(u)+1\}} \quad (1)$$

In (1) $f_q(s)$ is the fitness of s *th* individual in a population *PopM*[q] (q=1,2, number of individual). While $g_q(u)$ is the makespan of u *th* individual when it cooperates with the cooperative partner from *PopO*. The equation rescaled the value of $g_q(s)$ so that it makes the selection more effective and deals with minimum problem.

C. Genetic component for machine selection

1) Initial population

In order to generate a promising initial population for the machine selection problem, we adopt two approaches presented by Pezzella et al.[2]. The first approach is *AssignmentRule1* (search for the global minimum in the processing table) and the second approach is *AssignmentRule2* (randomly permute jobs and machine in the processing table). These two approaches are the modified version of the approach of localization by Kacem et al.[7]. In the initial population 10% of the individual is generated by *AssignmentRule1* and 90% of the individual is generated by *AssignmentRule2.*

2) Individual representation

Parallel job representation (PJ_r) was used as the individual representation in the machine selection problem. PJ_r is first introduced by Meshouni et al.[14]. This PJ_r is represented in a matrix form, where each row of the matrix is an ordered series of operation for each job. Meanwhile, each cell of the matrix consists of assigned machine and the starting time of the job operation. Moreover, this representation allows both row crossover and column crossover to be easily performed. But it needs a repairing mechanism to recalculate the starting time for every job operation after performing the genetic operation. Thus, we have improved this PJ_r with some modifications to avoid production of infeasible solution. In our approach, each cell of PJ_r only consists of the assigned machine and this machine is selected from the assignment rule.

TABLE II PARALLEL JOB REPRESENTATION

	O_1	O_2	O_3
J_1	M_1	M_4	-
J_2	M_2	M_3	-
J_3	M_5	M_4	M_4
J_4	M_3	M_1	M_2

3) Genetic operation

There are two crossover operator used to produce a new offspring. Row crossover and column crossover adopted from[14] are used in our approach as these operators always produce a legal offspring. The algorithm for the row crossover is given as:

Step1. Two individuals are randomly selected by the roulette wheel selection. One job (row of matrix) J is randomly selected.

Step2. The assigned machine for the selected job remains unchanged for both selected individuals.

Step3. Swap the assigned machine for the remaining job for both individuals.

The step for column crossover is similar to the row crossover, but the column crossover is swapped by the selected operation (column of the matrix) for the selected individual.

Mutation operator only changed the machine assignment properties of the individual. In Fig. 1, an example of mutation operation performed on the PJ_r. Firstly a job J_4 and operation O_3 are selected to perform the mutation, and the currently assigned machine is M_4. Secondly, a machine randomly selected from an alternative machine set is $M = \{M_1, M_2, M_3, M_4, M_5\}$ (refer to TABLE I ninth row and third column). M_5 is the selected and assigned machine for its changes are J_4, O_3 to M_5

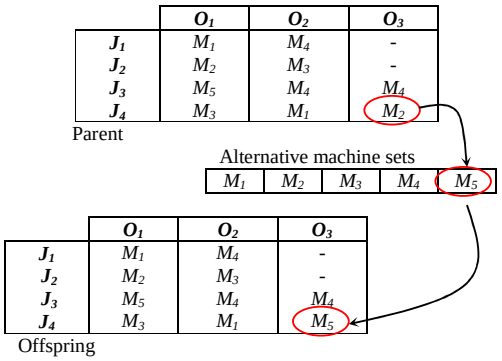

Fig. 1 Mutation for parallel job representation

D. Genetic component for operation sequence

1) Initial population

Initial population of the operation sequence is obtained by sequencing the operation on machine based on the initial population of machine selection. Initial population is generated from the mixing of three well know dispatching rules such as the most work remaining (MWR), most operation remaining (MOR) and random select job (RSJ). There are 40% of individual generated by MWR, 40% of individual generated by MOR and 20% of individual is generated by RSJ.

2) Individual representation

Operation sequence representation is used to encode the operation sequencing problem. In this representation all operations for the same job are defined with a same symbol and it interprets them according to the order. Thus, infeasible solution can be avoided by using the same symbol for the same job. The chromosome length L is the total operations of all jobs. An example of the operation sequence representation is constructed based on TABLE I. In TABLE I job J_1 consists of two operations (O_{11} - O_{12}) and job J_2 consists of two operations (O_{21} - O_{22}). However for job J_3 and J_4 each of this job consist of three operations. The operations for J_3 are ($O_{31,}$-O_{32}-O_{33}) and J_4 are ($O_{41,}$-O_{42}-O_{43}). In Fig. 2 a chromosome that contains of 2-1-3-4-4-2-3-1-4-3 is constructed. This data is read from left to right and there is an increasing operation index for each job. Thus it can be translated into O_{21}- O_{11}- O_{31}- O_{41}- O_{42}- O_{22}- O_{32}- O_{12}- O_{43}- O_{33}.

Operation Sequence representation

2	1	3	4	4	2	3	1	4	3

Fig. 2 Operation sequence representation

3) Genetic operator

In applying the crossover operator for the representation of operation the sequencing precedence of constraint among job must not be violated because it might be producing an illegal offspring, and to repair an illegal offspring is very time

consuming. Therefore the precedence order crossover (POX) operator from Lee et al.[18] concerns on the precedence order is adopted. The POX works as follows:

Step 1 Generate two sub-job set J_{s1}/J_{s2} from all jobs and select two parent individuals as O_{s1} and O_{s2} randomly;

Step 2 Copy any element in O_{s1}/O_{s2} that belongs to J_{s1}/J_{s2} into the child individual $O_{s'1}/O_{s'2}$, and retain the same position in $O_{s'1}/O_{s'2}$;

Step 3 Delete the elements that are already in the sub-job J_{s1}/J_{s2} from O_{s1}/O_{s2};

Step 4 Orderly fills the empty position in $O_{s'1}/O_{s'2}$ with the remainder elements of O_{s2}/O_{s1};

In Fig. 3, we used chromosome representation that consists of 4 jobs to demonstrate the procedure of POX. First two sub jobs are generated $J_{s1} = \{2,3\}$, $J_{s2}=\{1,4\}$. After that, copy the element in O_{s1} which belongs to J_{s1} into the child $O_{s1'}$ and, remain in the same position. And it is followed by deleting the element that is already in sub-job J_{s1} from O_{s2}. Lastly, fill the empty position in $O_{s1'}$ with the remainder elements of O_{s2}.

Swap mutation is applied to the operation sequence representation. First, two positions of the chromosome are randomly selected. Secondly, swap the element of the selected position.

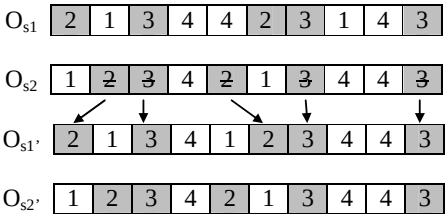

Fig. 3 Procedure of precedence order crossover

IV. Proposed Algorithm

The procedure for CCGA for FJSP is given as below:

Step 1: *Initialization.* Generate initial population for machine selection population *PopM[k], k = {1,2…N}*, and operation sequence population *PopO[q], q = {1,2...N}.*

Step 2: *Initial fitness evaluation*. Each individual from *PopM[k]*and *PopO[q]* is evaluated by combining them with the cooperative partner and set the f_{best} to be the fitness value of the individual. Cooperative partner is randomly selected from other species to form a complete solution.

Step 3: *Co-evolution*

Step 3.1 Select two parents from the population based on the fitness by the roulette wheel selection and applied crossover operator to generate two new offspring

Step 3.2 Mutation operator is applied to obtain new offspring

Step 3.3 Evaluate the fitness value of the new offspring by combining it with the cooperative partner. The random individual, best individual from previous evaluation and individual who stay at the same position is being evaluated and is selected as the cooperative partners.

Step 3.4 Set m←m+1.
If m<M (the number of species), then go to Step 3.1. Otherwise go to Step 4.

Step 4 *Termination*. If the termination criteria are satisfied, then the process will be stopped. Otherwise go to Step 3.

Fig. 4 is shown to explain the algorithm framework clearly. In Fig. 4, a co-evolutionary model of two species on this study is given. It denotes the evolution process for each species from the perspective of each in turn.

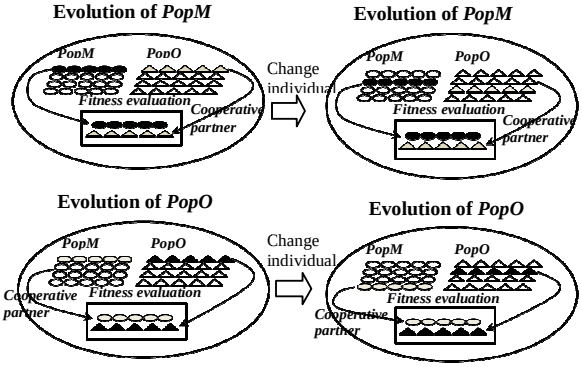

Fig. 4 Cooperative co-evolutionary genetic algorithm for FJSP

V. Computational Result

The proposed algorithm was implemented in java on an Intel Core 2 Duo running at 2.40 GHz, and tested on 10 dataset from Brandimarte[11]. This data set can be obtained from http://www.idsia.ch/~monaldo/fjsp.html/. The job number of the dataset is in the range of 10 to 20, the number of machine is in the range of 4 to 15 and the number of operations is in the range of 5 to 10. In order to evaluate the efficiency, the proposed algorithm was to test 5 times on each for every problem instance. The genetic parameter used is given as:

TABLE III COMPARISON OF CCGA WITH OTHER APPROACHES

Problem	$n \times m$	Flex	UB	CCGA		GENACE[1]			GA[2]			hGA[4]		
				PopSize	C_M	*PopSize*	C_M	*dev %*	*PopSize*	C_M	*dev %*	*PopSize*	C_M	*dev %*
MK01	10 × 6	2.09	(36,42)	2000	41	100	40	-2.50	5000	40	-2.50	3000	40	-2.50
MK02	10 × 6	4.1	(24,32)	1000	27	100	29	+6.90	5000	26	-3.85	3000	26	-3.85
MK03	15 × 8	3.01	(204,211)	500	204	100	N/A	N/A	5000	204	0.00	2000	204	0.00
MK04	15 × 8	1.91	(48,81)	2500	62	100	67	+7.46	5000	60	-3.33	3000	60	-3.33
MK05	15 × 4	1.71	(168,186)	1000	173	100	176	+1.70	5000	173	0	1000	172	-0.58
MK06	10 × 15	3.27	(33,86)	2000	64	100	67	+4.48	5000	63	-1.59	2000	58	-10.34
MK07	20 × 5	2.83	(133,157)	1000	140	100	147	+4.76	5000	139	-0.72	1000	139	-0.72
MK08	20 × 10	1.43	523	500	523	100	523	0.00	5000	523	0.00	1000	523	0.00
MK09	20 × 10	2.53	(299,369)	1500	328	100	320	-2.50	5000	311	-5.47	1000	307	-6.84
MK10	20 × 15	2.98	(165,296)	2000	225	100	229	+1.75	5000	212	-6.13	2000	197	-14.21
Average improvement								+2.21			-2.36			-4.24

- Population size: 500 to 2500
- Number of generation: 1000
- Crossover rates: 0.7
- Mutation rates: 0.01

The computational result is compared with the GENACE from Ho and Tay[1], GA from Pezzella et al.[2] and hybrid GA from Yadzni et al.[9] is given in TABLE III. In column 2 $n \times m$ denotes the number of jobs × number of machines, and *Flex* in column 3 denotes the average number of machine for each operation while the *Pop Size* represents the population

size used for every problem, the population size depends on the complexity of the problem (eg. number of machine, number of job, flexibility of the problem).The best makespan obtained from our CCGA after 5 runs of experiment is denoted as C_M in column 6. The best makespan from the GENACE, GA and hGA is represented in column 8,11 and 14 respectively. Moreover, the relative deviation is defined as dev = [(C_M(CCGA) - C_M(Comp)) / C_M(Comp)]*100%. C_M(Comp) is the makespan that we compared to, while C_M(CCGA) is the makespan obtained from our CCGA. The 13th row in the table denotes the average improvement of our CCGA compared with other approaches.

The *PopSize* (population size) used for our CCGA is smaller compared to GA[2] and hGA. Although the population size used for GENACE is 100 and it is smaller compared to the population size of our CCGA, but the makespan obtained by our CCGA has outperformed the GENACE[1].

In order to compare the performance of our CCGA and GA, we developed a GA that uses the same genetic operation. However the chromosome representation used in GA consists of two parts that are the machine selection (parallel job representation) and operation sequence (operation sequence representation). Besides that, the genetic parameter used to test GA is the similar with parameter stated in Section V. In Fig 5, comparison on the evolution process for GA and CCGA on benchmark problem Mk10 is given. In Fig 5, CCGA obtains makespan of 225 in the 313th generation. However at the 313th generation the makespan obtained by GA is 270. GA takes long time (more generation) to obtain the minimum makespan. Thus, we can conclude that CCGA improve the convergence speed and it optimum the searching ability compared to GA.

VI. CONCLUSION

Cooperative co-evolutionary genetic algorithm is proposed to solve the FJSP problem in this study. The computational result in TABLE III indicates that our CCGA is comparable with other approaches. In CCGA, the FJSP is divided into two species based on its difficulties and each of these species is maintained in a population. Furthermore by maintaining different species in different population, the population does not converge to a single individual. Besides that, each population evolves by a standard genetic algorithm. From Fig. 5 in section V, it can be observed that adopting two parallel searches in two small problems is more efficient compared to a single search in a big problem. Besides that, CCGA speeds up the convergence. In future the technique in this study can be applied in other areas such as project planning management, and transportation scheduling problem.

Evolution process for CCGA and GA on Mk10

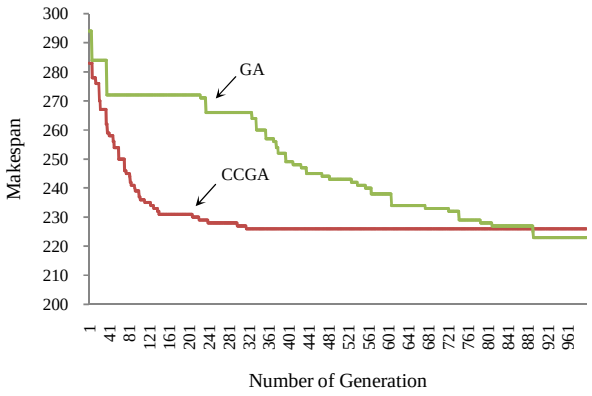

Fig. 5 Comparison on evolution process for CCGA and GA

REFERENCES

[1] N.B. Ho and J.C. Tay. GENACE: an efficient cultural algorithm for solving the flexible job-shop problem. in *Evolutionary Computation, 2004. CEC2004. Congress on*. 2004: p. 1759-1766 Vol.2F. Pezzella, G. Morganti, and G. Ciaschetti, *A genetic algorithm for the Flexible Job-shop Scheduling Problem.* Comput. Oper. Res., 2008. 35(10): p. 3202-3212.

[2] F. Pezzella, G. Morganti and G. Ciaschetti, A genetic algorithm for the Flexible Job-shop Scheduling Problem. *Comput. Oper. Res.*, 2008. **35**(10): p. 3202-3212J. Gao, L. Sun, and M. Gen, *A hybrid genetic and*

variable neighborhood descent algorithm for flexible job shop scheduling problems. Computers & Operations Research, 2008. **35**(9): p. 2892-2907.

[3] P. Brucker and R. Schlie, Job-shop scheduling with multi-purpose machines. *Computing*, 1990. 45(4): p. 369-375.S. Dauzère-Pérès and J. Paulli, *An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search.* Annals of Operations Research, 1997. 70(0): p. 281-306.

[4] J. Gao, L. Sun and M. Gen, A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research*, 2008. 35(9): p. 2892-2907.

[5] H. Chen, J. Ihlow and C. Lehmann. A genetic algorithm for flexible job-shop scheduling. in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. 1999: p. 1120-1125 vol.

[6] S. Dauzère-Pérès and J. Paulli, An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 1997. 70(0): p. 281-306.

[7] I. Kacem, S. Hammadi and P. Borne, Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 2002. **32**(1): p. 1-13.

[8] L.-N. Xing, Y.-W. Chen and K.-W. Yang, Multi-population interactive coevolutionary algorithm for flexible job shop scheduling problems. *Computational Optimization and Applications*.

[9] M. Yazdani, M. Amiri and M. Zandieh, Flexible job-shop scheduling with parallel variable neighborhood search algorithm. *Expert Syst. Appl.*, 2010. 37(1): p. 678-687.M. Mastrolilli and L. M. Gambardella, *Effective Neighborhood Functions for the Flexible Job Shop Problem*. 1998, Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale.

[10] M. Yazdani, M. Gholami, M. Zandieh and M. Mousakhani, A Simulated Annealing Algorithm for Flexible Job-Shop Scheduling Problem. *Journal of Applied Sciences*, 2009. 9(4): p. 662-670.

[11] P. Brandimarte, Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 1993. 41(3): p. 157-183.

[12] J. Hurink, B. Jurisch and M. Thole, Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spectrum*, 1994. 15(4): p. 205-215.K. A. De Jong and M. A. Potter, *A Cooperative Coevolutionary Approach to Function Optimization*, in *The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature*. 1994, Springer-Verlag. p. 249-257.

[13] M. Mastrolilli and L.M. Gambardella, *Effective Neighborhood Functions for the Flexible Job Shop Problem.* 1998, Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale.

[14] Mesghouni K., Hammadi S. and Borne P. Evolution programs for job-shop scheduling. in *Systems, Man, and Cybernetics, 1997. 'Computational Cybernetics and Simulation'., 1997 IEEE International Conference on*. 1997: p. 720-725 vol.1

[15] E.M. David and M. Risto, Forming neural networks through efficient and adaptive coevolution. *Evol. Comput.*, 1997. 5(4): p. 373-399.

[16] K.A. De Jong and M.A. Potter, Evolving Complex Structures via Cooperative Coevolution. *Evolutionary Programming*, 1995.

[17] K.A. De Jong and M.A. Potter, *A Cooperative Coevolutionary Approach to Function Optimization*, in *The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature*. 1994, Springer-Verlag. p. 249-257.

[18] K.M. Lee, T. Yamakawa and L. Keon-Myung. A genetic algorithm for general machine scheduling problems. in *Knowledge-Based Intelligent Electronic Systems, 1998. Proceedings KES '98. 1998 Second International Conference on*. 1998: p. 60-66 vol.2

Ms. Lee Yih Rou received her B.Sc. degree in Universiti Teknologi Malaysia, Malaysia in 2009. She is currently pursuing her Master degree in Faculty of Computer Science and Information System, Universiti Teknologi Malaysia. Her current research interest includes machine scheduling and artificial intelligence.

Dr. Hishammuddin Asmuni, is serving as a lecturer at the Software Engineering Department, Faculty of Computer Science and Information Systems, Universiti Teknologi Malaysia. He received his degree in Computer Science at the Universiti Malaya in 1996. He received MSc degree in Computer Science from Univeriti Teknologi Malaysia in 1999 specializing in the area of Software Engineering. After teaching for four years, he pursued his Ph.D. at The University of Nottingham, United Kingdom and received Ph.D. in Computer Science specializing in the area of Artificial Intelligence in 2008. His current research interest includes optimization techniques particularly in parallel bio-inspired algorithm.