Journal of the Korean Institute of Industrial Engineers
Vol. 40, No. 6, pp. 642-647, December 2014.
ISSN 1225-0988 | EISSN 2234-6457

http://dx.doi.org/10.7232/JKIIE.2014.40.6.642
© 2014 KIIE
<Original Research Paper>

Minimizing the Total Stretch in Flow Shop Scheduling with Limited Capacity Buffers

Suk-Hun Yoon†

Department of Industrial and Information Systems Engineering, Soongsil University

한정된 크기의 버퍼가 있는 흐름 공정 일정계획의 스트레치 최소화

윤 석 훈

숭실대학교 산업정보시스템공학과

In this paper, a hybrid genetic algorithm (HGA) approach is proposed for an n-job, m-machine flow shop scheduling problem with limited capacity buffers with blocking in which the objective is to minimize the total stretch. The stretch of a job is the ratio of the amount of time the job spent before its completion to its processing time. HGA adopts the idea of seed selection and development in order to improve the exploitation and exploration power of genetic algorithms (GAs). Extensive computational experiments have been conducted to compare the performance of HGA with that of GA.

Keywords: Scheduling, Flow Shop, Stretch, Genetic Algorithms

1. Introduction

The flow manufacturing line have been considered as a basic device for modern automated production systems and modeled by flow shop scheduling. The recent survey has pointed out the significance of no-wait and limited capacity buffers in modern systems such as just-in-time system and flexible manufacturing system (Liu *et al*., 2007).

Nowicki (1999) deals with the makespan minimization for the permutation flow shop problem with limited capacity buffers. He proposes a local search technique using a non-trivial generalization of the block elimination properties. Smutnicki (1998) presents a tabu search based approximation algorithm for the makespan minimization flow shop scheduling problem in a two machine with finite capacity buffers. Liu *et al.* (2007) propose a hybrid algorithm based on particle swarm optimization for the permutation flow shop scheduling problem with the limited buffers to minimize the makespan. Qian *et al*. (2009) propose a hybrid algorithm based on differential evolution for permutation flow shop scheduling with limited capacity buffers to minimize the makespan. Agnetis *et al.* (1997) deal with the makespan minimization problem in a two machine flow shop with limited capacity buffer. They show that in some conditions of a batch size, the problem can be formulated as a special case of the traveling salesman problem and solved in polynomial time. Pranzo (2004) investigates the makespan minimization problem in a two machine flow shop with limited capacity buffer and sequence independent setup times and removal times. He shows that in some conditions on the batch sizes, the problem can be solved in polynomial time by defining a suitable instance of no-wait flow shop. Norman (1999) investigates the flow shop scheduling with sequence dependent setup times and finite buffers, in which the objective is to minimize the makespan. He proposes a tabu search based solution procedure. Wang and Tang

† Corresponding author : Professor Suk-Hun Yoon, Department of Industrial and Information Systems Engineering, Soongsil University, Seoul 156-743, Korea, Tel : +82-2-820-0687, Fax : +82-2-825-1094, E-mail : yoon@ssu.ac.kr

Received November 11, 2014; Accepted December 1, 2014.

(2007) investigate the hybrid flow shop scheduling with finite intermediate buffers, whose objective is to minimize the total weighted completion times. They propose a tabu search heuristic based on the scatter search mechanism. Wardono and Fathi (2004) investigate the scheduling problem on parallel machines in successive stages with limited buffer capacities. They propose a tabu search algorithm in which the search is limited to the space of permutation vectors.

Traditionally, the focus of performance measure has been on the maximal completion time and the flow time. Recently, alternative performance measures have been considered and among them, the stretch measure has received a lot of attention (Chan *et al.*, 2006). The stretch of a job formally is defined as the ratio of its flow time to its required processing time (Bender *et al.*, 2004). The stretch measure relates the jobs' waiting times to their processing times, and may reflect users' psychological expectation that, in a system with highly various job sizes, users are willing to wait longer for larger jobs (Muthukrishnan *et al.*, 2005).

Since the problem of minimizing the total stretch with different job release times is not tractable even for a single machine, the computational effort to solve an instance of a problem grows remarkably quickly as the number of jobs increases (Pinedo, 2012). Thus, if the number of jobs is large, it is necessary to consider meta-heuristic methods such as genetic algorithms, simulated annealing, and tabu search (Dreo *et al.*, 2005).

This paper presents a solution methodology for an *n*-job, *m*-machine flow shop scheduling problem with limited capacity buffers in which the objective is to minimize the total stretch. In the next section, the problem and notations are defined. By adopting new genetic operators (seed selection and development), HGA is proposed to solve the problem in section 3. In section 4, the results of extensive computational experiments comparing HGA and GA are provided. Finally, a summary of main results and conclusions are provided in Section 5.

2. Minimization of the Total Stretch in the *n*-Job, *m*-Machine Flow Shop with Limited Capacity Buffers

For job j, $j = 1, \cdots, n$, let r_j be the release time, p_{ij} the processing time on machine i, $i = 1, \cdots, m$. Let p_j be the sum of processing times of job j ($= p_{1j} + p_{2j} + \cdots + p_{mj}$). If the completion time of job j on machine i is c_{ij}, then the stretch $s_j = (c_{mj} - r_j)/p_j$. Let individual $\sigma_l = \{\sigma(1), \sigma(2), \cdots, \sigma(n)\}$, $l = 1, \cdots, w$, represents a job sequence where $\sigma(k) = j$, k = 1, $\cdots$, n, implies that job j is positioned kth in the sequence. Let $c_{i,\sigma(k)}$ represent the completion time of the kth job on machine i for individual σ_l. Let b_i be the size (capacity) of the buffer between machines i and i+1; that is, b_i is the maximum number of jobs that can be placed in the buffer at any time.

In a flow shop with blocking, a machine can process a job even if the downstream buffer is full. When the processing of the job finishes, it is required to wait in the machine's working area until the buffer is no longer full. The blocked machine can start processing another job when the downstream buffer releases at least one job to the next machine. Let $a_{i,j}$ be the starting time of job j on machine i (that is, $a_{i,j} = c_{i,j} - p_{i,j}$). Job $\sigma(k) = j_1$ can start its processing on machine i if job $\sigma(k - b_i - 1) = j_2$ starts its processing on machine i+1; that is, $a_{i,\sigma(k)} \geq a_{i+1,\sigma(k-b_i-1)}$. Equivalently, this can be represented as follows:

$$c_{i,\sigma(k)} - c_{i+1,\sigma(k-b_i-1)} \geq p_{i,\sigma(k)} - p_{i+1,\sigma(k-b_i-1)},$$
$$i = 1, \cdots, m-1,\ k = b_i + 2, \cdots, n.$$

For a given sequence σ, the problem can be formulated as follows :

$$\text{minimize } z(\sigma) = \sum_{j=1}^{n} \frac{c_{mj} - r_j}{p_j}; \tag{1}$$

subject to

$$c_{i+1,\sigma(k-b_i-1)} - c_{i,\sigma(k)} \leq p_{i+1,\sigma(k-b_i-1)} - p_{i,\sigma(k)},$$
$$\text{for } i = 1, \cdots, m-1, k = b_i + 2, \cdots, n; \tag{2}$$

$$c_{i,\sigma(k)} - p_{i,\sigma(k)} \geq c_{i,\sigma(k-1)},$$
$$\text{for } i = 1, \cdots, m, k = 2, \cdots, n; \tag{3}$$

$$c_{i,j} - p_{i,j} \geq c_{i-1,j},$$
$$\text{for } i = 2, \cdots, m, j = 1, \cdots, n; \tag{4}$$

$$c_{i,j} - p_{i,j} \geq r_j,$$
$$\text{for } i = 1, \cdots, m, j = 1, \cdots, n; \tag{5}$$

Constraint set (2) provides the relationships between jobs on a machine and the downstream machine when the downstream buffer is full. Constraint set (3) establishes the relationships between completion times of adjacent jobs on each machine and assures that a machine can process at most one job at the same time. Constraint set (4) insures that each job on the current machine cannot be transferred to the next machine before its processing is finished. Constraint set (5) states that all jobs are available at their release time.

3. Hybrid GA Approach

GAs are stochastic search methods designed to search large and complex spaces by exploitation of solutions and a robust exploration of the space (Lee *et al.*, 1997). GAs start with a collection (population) of solutions (individuals). With the survival of the fittest philosophy, GAs select individuals in a population to form a mating pool according to their fitness values. Individuals in the mating pool are randomly mated to become couples and each couple goes through the crossover and mutation process to produce two new solutions (offspring). The set of all these new born offspring becomes the population of the next iteration (generation). The population size remains fixed in all generations and the process continues until a predetermined termination criterion is satisfied (Bhattacharyya, 1999).

HGA uses a permutation representation for individuals, where a sequence of *n* jobs is defined by a permutation of integers $\{1,\cdots,n\}$. Most individuals in an initial population are generated randomly. The rest of the initial population are generated by the seed selection process, which is based on two steps: firstly, rules to generate the initial sequences such as the shortest release time, the shortest processing time on the first machine, and the shortest total processing times, and secondly, neighborhood search mechanisms such as the non-adjacent pairwise interchange, the extraction and forward shifted reinsertion, and the extraction and backward shifted reinsertion. Balancing the ratio of individuals of these two types enhances robust exploitation and exploration of good quality of solutions.

Mostly used methods to obtain fitness values of individuals in the literature are fitness value by scale and fitness value by rank (Liepins and Hilliard, 1989). Fitness value by scale (f_{scale}) can be obtained as follows :

$$f_{scale}(\sigma_l) = (z_{\max} - z(\sigma_l) + z_{\min})/z_{avg}, \quad \text{for } l = 1,\cdots,w,$$

where z_{max}, z_{min} and z_{avg} are the maximum, minimum and average objective values in the current population, respectively, w is the population size, and $z(\sigma_l)$ is the objective value of individual l. Fitness value by rank (f_{rank}) can be obtained as follows :

$$f_{rank}([\sigma_l]) = 2l/w(w+1), \qquad \text{for } l = 1,\cdots,w,$$

where $[\sigma_l]$ is the l^{th} individual in a descending order of objective function values.

To determine the expected number of copies of individual l for a mating pool, $E(l)$, HGA uses the stochastic remainder selection procedure without replacement. $E(l)$ can be calculated as follows :

$$E(l) = wf(\sigma_l)/\sum_{j=1}^{w} f(\sigma_j),$$

where $f(\sigma_l)$ is the fitness value of individual l. $\lfloor E(l) \rfloor$ copies of individual l, $l = 1,\cdots,w$, are assigned to the mating pool. If $E(l)$ is not integer, Bernoulli trials with success probabilities $P_s(l) = E(l) - \lfloor E(l) \rfloor$ are performed to individual l one by one until the mating pool is full. When individual l is selected, $P_s(l)$ is reduced to 0.

Individuals in a mating pool are mated randomly and go through crossover and mutation process. HGA uses the partially matched crossover (PMX) for crossover (Goldberg, 1989). Two crossover points are picked at random and the genes between these two points are swapped and used to construct a match table. The genes before the first crossover point and after the second one are exchanged if the genes are included in the table. The process to construct the table and swapping genes between two individuals are explained below. For example, suppose that *A* and *B* are the individuals chosen for crossover such that *A* = (8 3 7 **1 2 6** 4 5) and *B* = (3 7 4 **2 8 5** 1 6), and the two crossover points are 3 and 6. First, the genes between two crossover points are swapped (1, 2, 6 of A and 2, 8, 5 of B). Second, the genes in the same positions between two crossover points are compared to construct the match table. The comparison ($1 \leftrightarrow 2$, $2 \leftrightarrow 8$, $6 \leftrightarrow 5$) results in the match table ($1 \leftrightarrow 8$, $6 \leftrightarrow 5$). According to the table, if genes have the value 1, 8, 6, and 5, they are exchanged into 8, 1, 5, and 6, respectively. Thus, the resulting individuals by PMX are *A'* = (1 3 7 2 8 5 4 6) and *B'* = (3 7 4 1 2 6 8 5). HGA adopts the adjacent swap method for the mutation process, in which a job is exchanged with the next job in the job sequence. If the last job is to be mutated, it is exchanged with the first job in the job sequence.

The stochastic remainder selection procedure without replacement limits the maximum number of copies of individual *l* in the mating pool and thus, high fit individuals cannot prevail in the early generations. However, using this selection scheme may increase the probability of selecting the least fit individuals in comparison with other selection schemes such as the roulette wheel selection method. The selection of the least fit individuals may provide low fit offspring and as a result, may decrease the search power for the best solutions. HGA applies a non-adjacent pairwise interchange (NAPI) method to the least fit individual in the mating pool, and replaces the least fit individual by the best individual in its neighborhood.

Hybrid Genetic Algorithm (HGA)

Step 0 (Initialization)

In a preliminary test, the best set of following parameters is determined before the main test : determination of fitness values (f_{scale}, f_{rank}), population size (w), number of generations (*GEN*), crossover probability (P_c), mutation probability (P_m), number of seed selection individuals (N_s)

Step 1 (Construction of an initial population)

(a) Generate w-N_s individuals using a random number enerator.
(b) Generate N_s individuals by seed selection

Step 2 (Evaluation and selection)

(a) Obtain objective values of individuals in the population
(b) Compute the fitness values of individuals in the population.
(c) Use the stochastic remainder sampling without replacement to select individuals from the population to form a mating pool.

Step 3 (Development)
Apply the NAPI method the least fit individual selected in Step 2(c). Replace the least fit individual by the best individual obtained by the NAPI.

Step 4 (Reproduction)
(a) Mate individuals in the mating pool randomly.
(b) Apply PMX to the couples.
(c) Apply the adjacent swap method to the offspring produced in (b).

Step 5 (Termination test)
If HGA reaches *GEN*, stop. Otherwise, go to Step 2.

4. Computational Study

The HGA and GA were coded in Visual FORTRAN and ran on an Intel Core i7 CPU@3.4 GHz PC. Since no sample problems were found in the literature that could be used as a benchmark for testing the proposed HGA, the test problems were generated randomly for buffers with 1, 2, and infinite capacity. Processing times and release times of jobs for the test problems were generated randomly according to the integer uniform distributions provided in <Table 1>.

The experiments were divided into two parts : a preliminary test and a main test. Since the performances of GA and HGA are influenced by several control parameters, a preliminary test is necessary to achieve the best parameter set for GA and HGA. In the preliminary test, 5 test problems of different sizes generated according to the data in <Table 1> were solved. The best average objective function value was obtained by using the fitness function by rank, a total of 10 seed individuals, a population size of 100, a total of 100 generations, a crossover rate of 1.0, and a mutation rate of 0.01.

The test problems for the main test were generated in a similar way. Eight different test problems were generated for each problem size. These 192 problems were solved by HGA. For small size flow shop problems (5 and 7 jobs and 2～5 machines), the results of HGA were compared with the optimal solutions obtained by exhaustive search. HGA achieved optimal solutions for all small size problems. HGA was applied to medium size (10 and 15 jobs and 2～5 machines) and large size (20 and 30 jobs and 2～5 machines) problems. To evaluate the performance of HGA, the solutions obtained by HGA were compared with the solutions provided by GA.

Table 1. Data used to generate test problems (all data are integers)

Data	Value
Number of jobs (NJ)	5, 7, 10, 15, 20, 30
Number of machines (NM)	2, 3, 4, 5
Job processing times on machines	Uniform (1, 31)
Job release times	Uniform (1, 6)

Table 2. Results for medium and large size, limited/infinite capacity buffer problems

(a) Buffer size of one

No. of Jobs	No. of machines	GA method	HGA method	% Dev
		Avg. obj. value (z_g)	Avg. obj. value (z_h)	$(z_g\text{-}z_h/z_g)\times100$
10	2	27.12	26.37	2.79
	3	22.73	22.26	2.06
	4	20.81	20.07	3.56
	5	19.28	18.64	3.33
15	2	54.85	52.03	5.15
	3	46.01	43.55	5.33
	4	41.36	37.54	9.22
	5	36.53	34.79	4.75
20	2	103.85	86.29	16.91
	3	78.24	71.99	7.98
	4	67.30	61.52	8.59
	5	59.43	56.09	5.61
30	2	229.42	189.98	17.19
	3	169.95	150.87	11.23
	4	146.37	129.10	11.80
	5	125.72	115.12	8.44
Average		78.06	69.76	7.75

(b) Buffer size of two

No. of Jobs	No. of machines	GA method	HGA method	% Dev
		Avg. obj. value (z_g)	Avg. obj. value (z_h)	$(z_g-z_h/z_g)\times100$
10	2	26.86	26.37	1.86
	3	23.14	22.34	3.28
	4	20.28	19.98	1.96
	5	19.19	18.59	3.55
15	2	56.57	51.91	8.47
	3	47.29	42.84	9.72
	4	40.35	37.04	7.21
	5	36.52	34.61	5.52
20	2	97.61	85.58	12.09
	3	79.29	71.98	8.92
	4	68.36	61.80	9.38
	5	59.82	55.69	7.07
30	2	227.67	187.59	23.64
	3	169.98	148.87	11.46
	4	144.32	127.93	10.37
	5	129.01	111.43	13.44
Average		77.89	69.03	8.38

(c) Buffer size of infinite

No. of Jobs	No. of machines	GA method	HGA method	% Dev
		Avg. obj. value (z_g)	Avg. obj. value (z_h)	$(z_g-z_h/z_g)\times100$
10	2	27.00	26.36	2.35
	3	23.45	22.11	5.73
	4	20.73	20.01	3.50
	5	19.15	18.59	2.92
15	2	56.58	51.81	8.42
	3	45.27	42.84	5.37
	4	39.63	36.86	6.99
	5	36.16	34.04	5.86
20	2	99.68	84.77	14.95
	3	80.67	71.90	10.88
	4	68.67	61.02	11.14
	5	61.39	55.80	9.11
30	2	227.75	183.10	19.60
	3	170.16	144.82	14.89
	4	145.65	124.38	14.60
	5	122.62	111.12	9.38
Average		77.78	68.10	9.11

The results of HGA and GA for medium and large size problems are shown in <Table 2>. The average objective function values reported in <Table 2> are the average values of eight instances for each problem size. Based on these results, HGA provides an 7.75, 8.38, and 9.11% improvement on the average for buffer sizes one, two, and infinite, respectively with comparison with GA .

5. Conclusions

In this paper, HGA has been proposed to prevent the premature convergence of GAs and maintain the search power by adopting the seed selection and the development process. By these new processes, HGA restrains the high fitness individuals from dominating populations in early generations. Extensive computational experiments have been conducted to compare the performance of HGA and that of GA. These results show that the average improvement of HGA over GA is 8.41% for different buffer size problems.

References

Agnetis, A., Pacciarelli, D., and Rossi, F. (1997), Batch scheduling in a two-machine flow shop with limited buffer, *Discrete Applied Mathematics*, **72**, 243-260.

Bender, M. A., Muthukrishnan, S., and Rajaraman, R. (2004), Approximation algorithms for average stretch scheduling, *Journal of Scheduling*, **7**, 195-222.

Bhattacharyya, S. (1999), Direct marketing performance modeling using genetic algorithms, *INFORMS Journal on Computing*, **11**, 248-257.

Chan, W.-T., Lan, T.-W., Liu, K.-S., and Wong, P. W. H. (2006), New resource augmentation analysis of the total stretch of SRPT and SJF in multiprocessor scheduling, *Theoretical Computer Science*, **359**, 430-439.

Dreo, J., Petrowski, A., Siarry, P., and Taillard, E. (2005), *Metaheuristics for Hard Optimization : Methods and Case Studies*, Springer, New York.

Goldberg, D. E. (1989), *Genetic Algorithms in Search*, *Optimization and Machine Learning*, Addison-Wesley, Reading, MA.

Lee, C.-Y., Piramuthu, S., and Tsai, Y.-K. (1997), Job shop scheduling with a genetic algorithm and machine learning, *International Journal of Production Research*, **35**, 1171-1191.

Liepins, G. E. and Hilliard, M. R. (1989), Genetic algorithms : Foundation and applications, *Annals of Operations Research*, **21**(1～4), 31-58.

Liu, B., Wang, L., and Jin, Y.-H. (1989), An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers, *Computers and Operations Research*, **35**, 2791-2806.

Muthukrishnan, S., Rajaraman, R., Shaheen, A., and Gehrke, J. F. (2005), Online scheduling to minimize average stretch, *Siam Journal on Computing*, **34**(2), 433-452.

Norman, B. A. (1999), Scheduling flowshops with finite buffers and sequence-dependent setup times, *Computers and Industrial Engineering*, **36**, 163-177.

Nowicki, E. (1999), The permutation flow shop with buffers : A tabu search approach, *European Journal of Operational Research*, **116**, 205-219.

Pinedo, M. L. (2012), *Scheduling : Theory, Algorithms, and Systems (4th Ed.)*, Springer, New York.

Pranzo, M. (2004), Batch scheduling in a two-machine flow shop with limited buffer and sequence independent setup times and removal times, *European Journal of Operational Research*, **153**, 581-592.

Quian, B., Wang, L., Huang, D. X., and Wang, X. (2009), An effective hybrid DE-based algorithm for flow shop scheduling with limited buffers, *International Journal of Production Research*, **47**(1), 1-24.

Smutnicki, C. (1998), A two-machine permutation flow shop scheduling problem with buffers, *OR Spektrum*, **20**, 229-235.

Wang, X. and Tang, L. (2009), A tabu search heuristic for the hybrid flowshop scheduling with finite intermediate buffers, *Computers and Operations Research*, **36**, 907-918.

Wardono, B. and Fathi, Y. (2004), A tabu search algorithm for the multistage parallel machine problem with limited buffer capacities, *European Journal of Operational Research*, **155**, 380-401.