Scientia Iranica, Vol. 9, No. 3, pp 276–282

SCIENTIA
IRANICA

Research Note

A Hybrid GA Model to Solve Regular and Block Flow-Shop Problems

M. Ghazanfari* and Z. Yaghoobi[1]

In this paper, a Hybrid Genetic Algorithm (HGA) model is presented to solve two kinds of flow-shop problem and minimize makespan. These problems are: Regular and block flow-shop problems. The flow-shop problem studied here is shown as $n/m/F/C_{\max}$. Genetic Algorithm (GA) technique is one of the efficient and robust techniques for solving combinatorial programming problems, which can be used either alone (Simple GA) or along with other techniques (Hybrid GA). The latter can benefit from other heuristics to reach the best solution more efficiently. The HGA model presented in this paper uses two heuristics developed already to solve the $n/m/F/C_{\max}$ problems. The Palmer and CDS heuristics are embedded in the model to generate the initial population. The model works based on elitism. The elitist strategy takes special care to preserve the best solutions and inserts them in the next generation. By analyzing the conducted experiments, it is found that the HGA is a powerful technique for solving flow-shop problems. The proposed GA model outperforms the well-known heuristics, both in terms of minimizing makespan and satisfying the closeness limitation in the block problems.

INTRODUCTION

Since Johnson published the first paper on flow-shop sequencing problems in 1954, this problem has held the attention of many researchers [1]. This problem is generally described as follows: There are m machines and n jobs, each job consists of m operations and each operation requires a different machine. These jobs have to be processed in the same sequence on m machines. The objective is to find the sequence of jobs minimizing the maximum flow time that is called makespan.

The main assumptions are usually made as follows:

- Every job has to be processed on all machines in the order $1, 2, 3, \cdots, m$;
- Every machine processes only one job at a time;
- Every job is processed on one machine at a time;
- The operations are not preemptable;
- The set-up times for the operations are sequence-independent and are included in the processing times;
- The operating sequences of the jobs are the same on every machine and the common sequence has to be determined.

There are $n!$ sequences for n jobs where just one, or a few of them, are the best solutions. However, investigating $n!$ cases is very difficult and time consuming. Hence, many heuristics have been developed that investigate only a few of these cases, though they do not provide necessarily optimum solutions.

HEURISTICS FOR GENERAL M-MACHINE PROBLEMS

Over the past three decades, extensive research has been done on the pure flow-shop problem. There is no easy algorithm which can provide an optimal solution. Integer programming and branch-and-bound technique can be used to find an optimal solution [2]. However, they are not very effective on large, or even medium-sized, problems. Hence, many heuristics have been developed, some more well-known of which are discussed below.

*. *Corresponding Author, Department of Industrial Engineering, Iran University of Science and Technology, Tehran, I.R. Iran.*

1. *Faculty of Technology, Science and Research Campus, Islamic Azad University, Tehran, I.R. Iran.*

Palmer Heuristic Algorithm

Palmer [3] proposed a slope order index to sequence the jobs on the machines, based on the processing time [3]. The idea is to give priority to jobs so that jobs with processing times that tend to increase from machine to machine will receive higher priority, while jobs with processing times that tend to decrease from machine to machine will receive lower priority. The slope index S_i for job i is calculated as:

$$S_i = \sum_{j=1}^{m}(2j - m - 1)t_{ij}, \quad i = 1, 2, \ldots, n \ . \tag{1}$$

Then, a permutation schedule is constructed by sequencing the jobs in nonincreasing order of S_i such as:

$$S_{i1} \geq S_{i2} \geq \ldots . \geq S_{in}. \tag{2}$$

Gupta Heuristic Algorithm

Gupta [4] suggested another heuristic which is similar to Palmer heuristic except that he defined his index in a different manner. The index S_i for job i is calculated as follows:

$$S_i = \frac{e_i}{\min\{t_{ik} + t_{i,k+1}\}}, \tag{3}$$

where:

$$e_i = \begin{Bmatrix} 1 & \text{if } t_{j1} < t_{jm} \\ -1 & \text{if } t_{j1} \geq t_{jm} \end{Bmatrix}. \tag{4}$$

Thereafter, the jobs are sequenced according to Index 3.

CDS Heuristic Algorithm

The Campbell, Dedek and Smith (CDS) heuristic [5] is basically an extension of Johnson algorithm. Its efficiency relies on two properties:

- Using Johnson rule in a heuristic fashion,
- Generally creating several schedules from which the best can be chosen.

This algorithm first generates a set of $m - 1$ two-machine problems by aggregating the m machines into two groups systematically. Then it applies Johnson two-machine algorithm to find the $m - 1$ schedules and finally, selects the best one among the schedules. At stage 1, consider the two-machine problem formed with machines 1 and m. At stage 2, consider the artificial two-machine problem. Artificial machine 1 is formed with machine group $\{1, 2\}$ and artificial machine 2 is formed with machine group $\{m, m - 1\}$. At stage k, consider the artificial two-machine problem: Artificial machine 1 with machine group $\{1, 2, \ldots, k\}$ and artificial machine 2 with machine group $\{m, \ldots, m - k + 1\}$. Aggregated processing times for the stage k are defined as follows:

$$t'_{i1} = \sum_{j=1}^{k} t_{ij} \quad \text{and} \quad t'_{i2} = \sum_{j=1}^{k} t_{i,m-j+1}. \tag{5}$$

The CDS heuristic has been found to be a very good and robust heuristic and it has been used in many studies as a standard of comparison.

NEH Heuristic Algorithm

The Nawaz, Enscore and Ham (NEH) heuristic algorithm [6] is based on the assumption that a job with a high total processing time on all the machines should be given higher priority than a job with a low total processing time. The NEH algorithm does not transform the original m-machine problem into one artificial two-machine problem. It builds the final sequence in a constructive way, adding a new job at each step and finding the best partial solution. The algorithm is as follows:

Step 1: Order the n jobs by decreasing the sums of processing time on the machines;

Step 2: Take the first two jobs and schedule them in order to minimize the partial makespan, as if there were only these two jobs;

Step 3: For $k = 3$ to n do: Insert the $k-$ job at the place, among the k possible ones, which minimizes the partial makespan.

GENETIC ALGORITHMS

Genetic algorithms have been successfully applied to solve flow-shop problems [7,8]. Gen, Tsujimura and Kulbota [9] developed a simple genetic algorithm (SGA) approach for flow-shop problems. They used the permutation of jobs as the representation scheme of chromosomes, which is a natural representation for a sequencing problem. Their fitness function for each chromosome is the inverse of makespan. They also used the well-known crossover operations available for permutation representation, such as PMX (partially mapped crossover), OX (order crossover) and CX (cycle crossover). Mutation is also designed to perform random exchange.

Some examples of the applications of genetic algorithms to job shop scheduling are [10-13]. Lee et al. [14], combine the strengths of genetic algorithms and induced trees, a machine learning technique, to develop a job shop scheduling system.

Reeves proposed a Hybrid Genetic Algorithm (HGA) method for solving flow-shop sequencing problems [8]. He hybridized the conventional heuristic in the initial generation to produce a good seed generation. One chromosome is generated with the NEH

heuristic algorithm, while the remaining chromosomes are generated randomly. Gen and Chang [1] compared the results of solutions between some well-known heuristic algorithms. They considered a simple example of five-job four-machine problem and concluded that the GA model outperforms other heuristics for small-sized problems. There are also further studied applications of GAs in flow-shop problems [15-18].

THE PROPOSED GA MODEL

In this paper, the Reeves idea for developing HGA is inspired to propose a new HGA model. The proposed model embeds the CDS and Palmer heuristics and benefits from their advantages to solve two kinds of problems. It is a sequel work for the model recently developed by Ghazanfari and Yaghoobi [19]. The problems, considered in this paper, are called as regular $n/m/F/C_{\max}$ and block $n/m/F/C_{\max}$ problems and described as follows:

1. Problem 1 (Regular): Minimizing the total makespan when there is no limitation on the closeness of jobs;
2. Problem 2 (Block): Minimizing the total makespan when there is a limitation on the closeness of jobs, i.e., some jobs must be close together. In this case, there is no limitation on the sequence of jobs, but some of them need to close as a block. Although the regular multi-job, multi-machine problems have been solved by different heuristics, less attention is paid to the block problem.

The proposed HGA uses the CDS and Palmer heuristics to generate the initial population. The initial population consists of the best individuals from these heuristics and, also, some individuals generated randomly. Applying the elitist strategy, the best individual from each population is transferred to the next population.

The Structure of the Model

There are a number of characteristics which form the structure of a GA model. These characteristics for the proposed GA model are as follows:

1. String: The permutation of jobs is used as the representation scheme of chromosomes. For example, let the kth chromosome be $V_k = [3\ 2\ 4\ 1]$. This means that the job sequence is j_3, j_2, j_4, j_1;
2. Evaluation function: This function denotes the makespan for a chromosome. A chromosome is better if its evaluation function is less than others;
3. Crossover operation: In these problems usually crossover operators, such as PMX (partially mapped crossover), OX (order crossover), CX (cycle crossover) multi-cut-point and one-cut-point crossover are considered. Here, the one-cut-point crossover is used. In the one-cut-point crossover, it picks a cross point randomly, takes the pre-cut section of the first parent and fills up the offspring by taking, in order, each legitimate gene from the second parent, as shown in Figure 1;
4. Mutation operator: A shift mutation is used, which is a shift of one gene (chosen randomly) to the right or left for a random number of places, as shown in Figure 2;
5. Selection: To select and copy chromosomes into the mating pool, they are sorted ascendingly with respect to their makespans. Then F_k number of them is copied to the mating pool. F_k is calculated as follows:

$$F_k = \text{int}\left(\sum_{i=1}^{n} c_i / n c_k\right), \tag{6}$$

where c_i is the makespan of ith chromosome and n is the size of the population. Using Equation 6, the best chromosome (with less makespan), sends the most copies into the mating pool.

6. Initial population: Two chromosomes are generated with Palmer and CDS heuristic algorithms and the remaining chromosomes are generated randomly. Note that generated chromosomes for Problem 2 (block problem) must satisfy the limita-

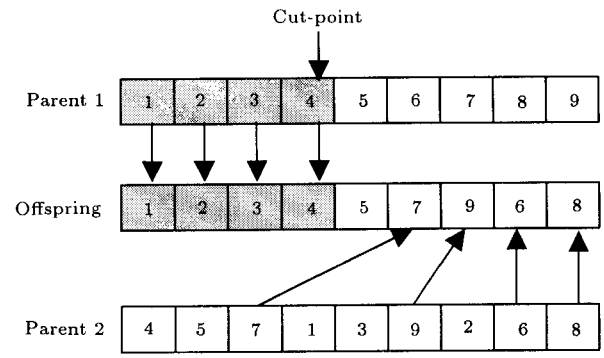

Figure 1. One-point crossover.

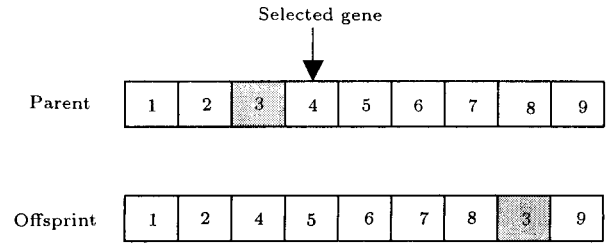

Figure 2. Shift mutation.

tion of closeness of pre-determined genes, otherwise the chromosomes are eliminated;

7. Criterion of stop: The criterion is a predefined number of generations;
8. Mating pool: It consists of the best individual or chromosomes as mentioned before. Better chromosomes have the greater number of copies in the mating pool. For each chromosome, the smaller makespan, the more copies in the mating pool;
9. Selection for crossover and mutation operations: Chromosomes are selected randomly for these operations;
10. Selection for the next population: The individuals (parents) in the mating pool are used to generate new chromosomes (offspring). Both of them (parents and offspring) are sorted ascendingly. From the top of this list, a predefined number of individuals are selected as the new generations. It should be noted that generated chromosomes for Problem 2 (block problem) must satisfy the limitation of closeness of some genes, otherwise the chromosomes are eliminated;
11. GA Parameters: These parameters are the initial population size, population size of other generations, the rate of crossover and mutation and the number of generations in a run. The user can define these parameters.

Pseudo-Code of the Proposed Model

This model has six steps. They are described as follows:

Step 0: In this step the system parameters are set;

Step 1: The solution of flow-shop problem is calculated by Palmer and CDS algorithms and considered as two elements of the initial population. Other elements are generated randomly. Makespans for all of the elements are then calculated;

Step 2: In this step, crossover operation is done N_{cross} times. For each crossover, three random numbers are generated. Two of them are used for selection of two members (as parents) and the last one is used to select the cross-point;

Step 3: In this step, a mutation operation is done. It operates N_{mut} times. First, three random numbers are generated. Then, using the first random number, an element is selected and, with two others, two places on this member are selected for shifting. Note that the generated chromosomes for Problem 2 (block problem) must satisfy the limitation of closeness of predetermined genes; otherwise the chromosomes generated in each of the above steps are eliminated;

Step 4: First, all offspring and all members of the last generation are put together. Then, they are listed ascendingly, based on their makespans. To form the new generation, a predefined number of members are selected from the top of this list. The iteration number is added by one, the number of copies of each member (for the mating pool) is calculated and the mating pool is formed again;

Step 5: If the iteration number is equal to the predefined number, the run is stopped; otherwise it goes to step 2.

The pseudo code of the proposed model is shown in Figure 3.

Program Code for the Model

A program code was written for this model in a FoxPro environment. The data can be provided in the program either from a prepared data file or input from a computer keyboard. The output of the program consists of the detailed data for the problem, generations and solutions from the Palmer and CDS heuristics and the proposed model.

NUMERAL EXPERIMENTS

To compare the performance of the proposed model with the well-known heuristic algorithms, eight different problems were solved. The results of these experiments are represented in Tables 1 and 2. Problem number 1 is a classic problem, taken from the literature.

The structure of Table 1 is as follows: Columns 1 and 2 indicate the number and the size of experiments respectively. Columns 3 to 8 show the results of solutions for Problem 1 (Regular) which is to minimize the total makespan when there is no limitation on the closeness of jobs. As seen, the proposed hybrid algorithm outperforms other heuristics in terms of makespan figures. Compared with an optimal solution, the hybrid solution gets very close to the optimal one.

Table 2 indicates the results for Problem 2 (Block), which aims to minimize the total makespan when there is a limitation on the closeness of jobs. It uses the same data test but assumes that jobs 1 and 2 must be close together. It this case, the proposed algorithm based on GA also provides near optimal solutions satisfying the above mentioned limitation.

DISCUSSION

During the conducting of a number of experiments, the following results were observed.

1. The initial population in HGA has an important role in finding the final solution. Hence, generating its

Table 1. Comparison of solution among the proposed algorithm, Palmer and CDS heuristics.

No	Problem Size	CDS	Palmer	SGA: Without Palmer & CDS		HGA: Using Palmer & CDS		Optimal Solution
	Job/Machine	Solution	Solution	Solution	Found in Pop. No.	Solution	Found in Pop. No.	
1	5/4	246	245	226	1	213	10	213
2	5/4	56	57	52	4	52	7	52
3	8/6	121	131	121	9	121	1	121
4	9/5	102	111	100	6	99	10	98
5	12/6	200	198	191	4	192	6	*
6	16/6	278	273	290	2	267	9	*
7	16/9	467	490	439	8	426	7	*
8	22/12	585	629	554	6	553	9	*

*Need more then 20 hours to calculate the optimal solution.

Table 2. The situation of solutions in regard to satisfying the limitation of closeness (Block Problem).

No	Problem Size	CDS	Palmer	The Proposed Hybrid GA Model		
	Job/Machine	Satisfying Limitation?	Satisfying Limitation?	Satisfying Limitation?	Solution Makespan	Found in Pop. No.
1	5/4	No	No	Yes	235	1
2	5/4	No	No	Yes	52	1
3	8/6	Yes	No	Yes	125	6
4	9/5	No	No	Yes	100	8
5	12/6	No	Yes	Yes	194	8
6	16/6	Yes	Yes	Yes	269	5
7	16/9	No	No	Yes	440	5
8	22/12	Yes	No	Yes	557	9

members by the Palmer and CDS heuristics, along with the random method, is useful;

2. Due to the elitist strategy applied in the model, there is no need to use a very big population size;
3. As the model embeds the Palmer and CDS heuristics, it always gives a better solution than these heuristics;
4. The advantages of the proposed model become clearer when it is used for problems of a large size;
5. Due to embedded heuristics and the status of initial population, the model reaches the best solution through less generation and less running;
6. The GA model produces alternative solutions with the same makespan. Therefore, when there is a difficulty in implementing a solution, other solutions can be applied;
7. While the well-known heuristics, such as CDS and Palmer, mostly fail to produce solutions for the block problem (Problem 2), the proposed GA model provides near-optimal solutions for this problem, satisfying the closeness condition.

CONCLUSION

The flow-shop sequencing problem is usually labeled as $n/m/F/C_{\max}$, which means n-job/m-machine/flow-shop/ maximum flow time. In this case, n jobs have to be processed in the same sequence on m machines. The objective is to find the sequence of jobs minimizing the maximum flow time, which is called makespan. Integer programming and branch-and-bound techniques can be used to find the optimal solution [2]. However, they have some disadvantages, e.g. they are not very effective on large or even medium- sized problems. It is also very time consuming, even impossible, to investigate

```
Step 0. System parameters
        M_init = 20 M = 20 N_cross = int(2/3M_mp + 1)
        N_mut = int(0.3M_mp + 1)  Iteration = 10
Step 1. Initial population
        generate (CDS-sequence)
        generate (Palmer-sequence)
        evaluate (CDS-sequence)
        evaluate (Palmer-sequence)
        pop - no ← 2 ;
        repeat
        pop - no ← pop - no +1
        generate (random - sequence)
        evaluate (random - sequence)
        until pop - no = M_init
        sort (population);
        calculate (population - copy - no)
        form (matting - pool)
        iter ← 1
Step 2. Crossover
        cross - no = 1
        repeat
        generate (random 1, random 2, random 3)
        select (parent 1, random 1)
        select (parent 2, random 2)
        crossover (parent 1, parent 2, cross-point)
        cross - no ← cross - no + 1
        until cross - no= N_cross
Step 3. Mutation
        mut - no = 1
        repeat
        generate (random 1, random 2, random 3)
        select (parent, random 1)
        select (mut-point 1, random 3)
        mutation (parent, mut - point 1, mut - point 2)
        mut - no ← mut - no +1
        until mut - no = N_mut
Step 4. Evaluation and selection
        evaluation (offspring - sequence)
        put (last - population) in temporary space
        insert (offspring - sequence) into temporary space
        sort (population) in temporary space
        select (M first) of temporary space for next population
        iter ← iter + 1
        calculate (population - copy - no)
        form (matting - pool)
Step 5. Stop checking
        if iter < iteration
          go step 2
        else
          stop
        end
```

Figure 3. Pseudo code of the proposed model.

all combinations to find the optimum solution. Hence, many heuristics, such as Palmer and CDS algorithms, have been developed to provide good, quick solutions, but cannot provide very good solutions for large size problems.

In this paper, a genetic algorithm model based on the hybrid approach is proposed. This model is designed to solve two kinds of problems, namely Problem 1 (Regular), which is for minimizing the total makespan when there is no limitation on the closeness of jobs and Problem 2 (Block), which is for minimizing the total makespan when there is a limitation on the closeness of jobs, i.e., some jobs must be close together. In this case, there is no limitation on the sequence of jobs, but some of them need to be adjacent jobs as a block.

The proposed model has better performance in terms of makespan value and in satisfying the closeness limitation in comparison with Palmer and CDS heuristics.

REFERENCES

1. Gen, M. and Cheng, R., *Genetic Algorithms and Engineering Design*, John Wiley & Sons, Inc (1997).
2. Ignal, E. and Schrage, L. "Application of the branch and bound technique to some flow-shop scheduling problems", *Operation Research*, **13**, pp 400-412 (1965).
3. Palmer, D. "Sequencing jobs through a multi-stage process in the minimum total time - A quick method of obtaining a near optimum", *Operation Research Quarterly*, **16**, pp 101-107 (1965).
4. Gupta, N.C., Gupta, Y.P. and Kumar, A. "Minimizing flow time variance in a single machine system using genetic algorithm", *European Journal of Operational Research*, **70**, pp 289-303 (1993).
5. Campbell, H., Duedeck, R. and Smith, M. "A heuristic algorithm for the n-job m-machine sequencing problem", *Management Science*, **168**, pp 630-637 (1970).
6. Nawaz, M., Enscore, E. and Ham, J. "A heuristic algorithm for the m-machine, n-job flow shop sequencing problem", *Omega*, **11**, pp 11-95 (1983).
7. Chen, C., Vempati, V. and Aljaber, N. "An application of genetic algorithms for flow-shop problems", *European Journal of Operational Research*, **80**, pp 389-396 (1995).
8. Reeves, C. "A genetic algorithm for flow-shop sequencing", *Computers and Operations Research*, **22**, pp 5-13 (1995).
9. Gen, M., Tsujimura, Y. and Kubota, E. "Solving job-shop scheduling problem using genetic algorithms" in *Proceedings of the 16th International Conference on Computer and Industrial Engineering*, M. Gen and T. Kobayashi, Eds., Ashikaga, Japan, pp 576-579 (1994).
10. Davis, L. "Job shop scheduling with genetic algorithms", *Proceedings of the International Conference on Genetic Algorithms and their Applications*, Grefenstette, J., Ed., pp 136-140 (1985).
11. Herrmann, J.W. and Lee, C.Y. "Solving a class scheduling problem with a genetic algorithm", *ORSA Journal of Computing*, **7**, pp 433-452 (1995).
12. Beam, J. "Genetic algorithms and random keys for sequencing and optimization", *ORSA Journal of Computing*, **6**(2), pp 154-160 (1994).
13. Ho, J. and Chang, Y. "A new heuristic for n-job m-machine flow-shop problem", *European Journal of Operational Research*, **52**, pp 194-202 (1991).

14. Lee, C.Y., Piramuthu, S. and Tsai, Y.K. "Job shop scheduling with a genetic algorithm and machine learning", *Inter. Journal of Production Research*, pp 131-139 (1997).

15. Chen, C. et al. "An application of GA for flow-shop problems", *European Journal of Operational Research*, **8**, pp 389-396 (1995).

16. Murata, T. et al. "GAs for flow-shop scheduling problem", *Comp. & IE*, **4130**, pp 1061-1071 (1995).

17. Murata, T. et al. "Multi-objective GA and its application to flow-shop scheduling" *Comp. & IE*, **4130**, pp 957-968 (1996).

18. Yagiura, M. et al. "The use of dynamic programming in GAs for permutation problems", *EJOR*, **92**, pp 387-401 (1996).

19. Ghazanfari, M. and Yaghoobi, Z. "A GA model to solve $n/m/F/C_{\max}$ flow-shop problems", *The Proceedings 4th Annual International Conference on IE Theory, Applications and Practice*, San Antonio, Texas, USA (1999).