Solving Single Machine Scheduling Problem with Maximum Lateness Using a Genetic Algorithm

Habibeh NAZIF (Corresponding author)

Department of Mathematics, Faculty of Science

Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, MALAYSIA

Tel: 603-8946-8454 E-mail: habibeh@math.upm.edu.my

Lai Soon LEE

Department of Mathematics, Faculty of Science

Laboratory of Applied and Computational Statistics, Institute for Mathematical Research

Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, MALAYSIA

Tel: 603-8946-8454 E-mail: lee@math.upm.edu.my

Abstract

We develop an optimised crossover operator designed by an undirected bipartite graph within a genetic algorithm for solving a single machine family scheduling problem, where jobs are partitioned into families and setup time is required between these families. The objective is to find a schedule which minimises the maximum lateness of the jobs in the presence of the sequence independent family setup times. The results showed that the proposed algorithm is generating better quality solutions compared to other variants of genetic algorithms.

Keywords: Genetic algorithm, Single machine scheduling

1. Introduction

Machine Scheduling Problems (MSPs) are motivated by the allocation of limited resources to jobs over time, subject to some constraints. In this study, we concentrat on a Single Machine Family Scheduling Problem (SMFSP) where jobs are partitioned into families and setup time is required between these families, with the objective of minimising the maximum lateness of the jobs in the presence of the sequence independent family setup times. (Hariri and Potts, 1997) described the problem as follows:

> "Given N jobs, each characterised by a processing time p_j, on a single machine, and a due date d_j, for $j = 1, 2, \ldots, N$, and a partition into F families. For each family f, for $f = 1, 2, \ldots, F$, jobs are split into batches, where a batch is defined as a maximal set of contiguously scheduled jobs from the same family which share the same setup. A sequence independent family setup time s_f, is required at the start of the schedule and also when there is a switch of jobs from another family. The objective is to find a schedule which minimises the maximum lateness $L_{\max}$, of the jobs in the presence of the family setup times."

The SMFSP for arbitrary family f is an NP-hard problem as shown by (Bruno and Downey, 1978). This problem can be represented as $1 \left| s_f \right| L_{\max}$ based on the standard classification of (Graham et al., 1979).

We review some approaches used for the problem of $1 \left| s_f \right| L_{\max}$. Excellent reviews on scheduling with setup considerations are given by (Potts and Kovalyov, 2000) and (Allahverdi et al., 2008).

(Monma and Potts, 1989) showed that there exists an optimal schedule in which the Earliest Due Date (EDD) rule of (Jackson, 1955) applies within each family f. Using dynamic programming, they solved the problems of $1 \left| s_{fg} \right| L_{\max}$ and $1 \left| s_f \right| L_{\max}$ in $O(F^2 N^{F^2+2F})$ and $O(F^2 N^{2F})$ time respectively. (Hariri and Potts, 1997) proposed a branch and bound algorithm. They obtained an initial lower bound by ignoring setups, except for those associated with the first job in each family, and solved the resulting problem with the EDD rule. They also designed two heuristics in which the first heuristic assigns all jobs of a family to a single batch, and the second heuristic splits each family into at most two batches according to the due dates of its jobs.

(Baker and Magazine, 2000) designed an algorithm that uses a branch and bound approach combined with dominance properties which reduced the effective problem size to solve the problem of $1 \left| s \right| L_{\max}$, where setup times are identical. (Zdrzałka, 1995) proposed two approximation algorithms without the unit setup time assumption and under non-positive due dates. (Pan et al., 2001) suggested a mathematical model that first finds initial schedule and then applies merging properties to improve the initial schedule.

(Lee et al., 2007) proposed a MultiCrossover Genetic Algorithm (MXGA) for $1\left|s_f\right|L_{\max}$. They hypothesised that generating multiple offspring during the crossover can improve the performance of a Genetic Algorithm (GA).

In the next section, we propose an Optimised Crossover Genetic Algorithm (OCGA) for the problem of $1\left|s_f\right|L_{\max}$. The computational results are presented and discussed in Section 3. A brief conclusion is given in the last section.

2. Optimised Crossover Genetic Algorithm (OCGA)

GA, originally developed by (Holland, 1975) is a adaptive heuristic search technique that mimics the natural evolution process. It works by combining biological processes such as selection, crossover and mutation operations. The selection pressure drives the population toward better solutions while crossover uses genes of selected parents to produce offspring that will form the next generation. Mutation is used to escape from local minima.

The widely used crossover operators involve some random choices with the result that a potentially optimal solution could be lost through incorrect choices being made in the randomisation process. (Aggarwal et al., 1997) proposed an optimised crossover mechanism for the independent set problem which takes into account the objective function in a straightforward way. They recognised that the merge operation formulated by (Balas and Niehaus, 1996) can be thought of as an optimised crossover. Hence, they constructed a bipartite graph from the two parent independent sets and produced the two new children: the O-child (Optimum child) and the E-child (Exploratory child). The O-child is constructed in such a way that has the best objective function value from a feasible set of children, while the E-child is constructed so as to maintain the diversity of the search space. (Balas and Niehaus, 1998) developed this approach to produce a superior genetic algorithm. (Ahuja et al., 2000) proposed a greedy genetic algorithm for the quadratic assignment problem. They used both path crossover and optimised crossover scheme in their algorithm.

In the remainder of this section, we describe the proposed optimised crossover and discuss how various steps of the GA are implemented for the OCGA. Briefly, the OCGA selects two parents from the population and generates two children by an optimised crossover mechanism which is designed using an undirected bipartite graph. A F-point swap operator is used to produce children when the optimised crossover is not applied to the parents. To maintain the diversity within the population, a binary mutation is randomly applied to each child. Moreover an elitism replacement scheme with filtration strategy is used to preserve good solutions and to avoid premature convergence. The general framework of OCGA can be shown as follow:

Algorithm OCGA:

begin

Initialise Population (randomly generated);
Fitness Evaluation;
repeat
Selection (probabilistic binary tournament selection);
Optimised Crossover;
***F*-point swap** (if the optimised crossover is not applied);
Mutation (binary mutation);
Fitness Evaluation;
Elitism replacement with **Filtration**;
until the end condition is satisfied;
return the fittest solution found;

end

2.1 Encoding scheme and selection mechanism

A natural way of coding the problem would be to represent each solution by a bit string. We use a binary $\{0, 1\}$ representation that is proposed by (Mason, 1992). Using this binary representation we defined the partition of families into batches, where '1' means the first job in a batch and '0' means a contiguously sequenced job in a batch. The length of the individual corresponds to the number of jobs N. After choosing the representation, we uniformly randomly generate an initial population which is of size 100 in our implementation using a random number generator. We assume that the size of population is kept constant throughout the process. Moreover we use a probabilistic binary tournament selection scheme to select individuals from the population to be the parents for the OCGA with a given selection probability $p_s = 0.75$.

2.2 Fitness Evaluation

The fitness function is used to evaluate the individuals which are introduced into the population. We define the fitness function using the property of EDD rule for batches developed by (Baker, 1999), in which there exists an optimal schedule

where the batches are sequenced in a non-decreasing order according to their due dates. As mentioned earlier, a batch is a maximum group of contiguously scheduled jobs within a family. Let d_{fj} and p_{fj} denote the due date and processing time of the jth job from family f which is identified as pair (f, j). Let $(f, h), \ldots, (f, k)$ be the jobs of an arbitrary batch b, then the batch due date δ_b is defined as follows:

$$\delta_b = \min_{j=h,\ldots,k} \{d_{fj} + q_{fj}\} \quad \text{where} \quad q_{fj} = \sum_{i=h}^{k} p_{fi} - (p_{fh} + \cdots + p_{fj}) \tag{1}$$

In an optimal schedule, the batches are sequenced in a non-decreasing order according to their batch due dates δ_b (i.e. $\delta_b \leq \delta_{b+1}$). A sequence independent family setup time s_f is added before the start of each batch and when there is a switch in the processing jobs from one family to jobs of another family. We define fitness function as the maximum lateness $L_{\max}$ of the schedule defined as $L_{max} = \max_j\{L_j\}$ where $L_j = C_j - d_j$ and C_j denotes the completion time of job j.

2.3 Optimised Crossover

During optimised crossover, we generate the two new children O-child and E-child from a pair of selected parents. We will now explain the optimised crossover strategy on determining O-child and E-child for the problem of $1 \left| s_f \right| L_{\max}$.

Step 1: Identify the parent with the least $L_{\max}$ as P_1 and select family within the P_1, which contains the job where $L_{\max}$ occurs as family f. Label another parent as P_2. Note that this family f will be used in both P_1 and P_2.

Step 2: Construct an undirected bipartite graph $G = (U \cup V, E)$ where $U = \{u_1, u_2, \ldots, u_n\}$ representing the jobs of family f, $V = \{v_1, v_1', v_2, v_2', \ldots, v_n, v_n'\}$ representing bit situation of the jobs of family f in both P_1 and P_2 (i.e. $v_i, v_i' \in \{1, 0\}$), and E representing the arc set in this graph in which, $\{u_j, v_i\}, \{u_j, v_i'\} \in E$ if and only if job j is represented with the bit situation v_i and v_i' respectively.

Step 3: Determine all the maximum matchings in graph G. Suppose that there are k jobs of family f that are represented with a different bit situation in the two parents. There will be exactly 2^k maximum matchings in graph G.

Step 4: Generate a temporary offspring from P_1 by replacing the bit situations of the jobs in family f which corresponds to one of the maximum matchings in graph G. Repeat the procedure for $2^k - 1$ times to generate 2^k temporary offspring. Note that one of the temporary offspring is exactly the same as P_1, thus removed.

Step 5: Select a temporary offspring with the least $L_{\max}$ among $2^k - 1$ temporary offspring as O-child.

Step 6: Generate E-child from P_2 by replacing the bit situations of the jobs in family f with $((f_{p_1} \cup f_{p_2}) \backslash f_{O-child}) \cup (f_{p_1} \cap f_{p_2})$, where f_i indicates family f of individual i.

Since the number of temporary offspring will increase exponentially with the number k, we restricted the maximum temporary offspring in graph G in every case to 2^5 even if the jobs that have a different bit situation in two parents are more than 5. In addition the proposed optimised crossover is applied based on a crossover probability, $p_c = 0.75$.

2.4 F-point Swap

Reproduction of parents is used when crossover operator is not applied to the selected parents in a standard GA. (Lee et al., 2007) used a swap operator instead of the exact duplicate of the parents in their MXGA. In our study, we use a F-point swap operator to produce two new offspring when the optimised crossover is not applied to the parents. This operator could be regarded as a giant mutation where the elements in the parent are randomly reassigned. The process of F-point swap operator is as follows:

Step 1: Select randomly a swap point within family $f = 1$ from a parent to form two sub-strings.

Step 2: Swap the position of the sub-strings (except the first job in the selected family) with the swap point as the point of exchange.

Step 3: Repeat **Step 1** and **Step 2** for each family f ($f = 2, 3, \ldots, F$).

The steps above are repeated for the second parent to create a second offspring.

2.5 Mutation

We use binary mutation operator in our OCGA. Binary mutation is applied randomly to each offspring individually that alters each gene from '1' to '0' or vice versa with a given mutation probability $p_m = 1/N$, where N is the number of jobs. Note that each gene can be selected to be mutated except the first gene in each family.

2.6 Replacement

Elitism replacement scheme is applied in this study as follows: both parent and offspring population are combined into a single population and sorted in a non-increasing order of their associated fitness value. Then, the first half of the combined population is selected as the individuals of the new population for the next generation. In order to avoid premature convergence and to add diversity to the new population, we use the filtration strategy proposed by (Lee et al., 2007) in which, identical individuals are identified from the new population and, they are removed and replaced by uniformly randomly generated new individuals. As the filtration strategy requires a certain amount of computational time, the procedure will only be invoked after every 50 generations.

3. Computational Experiments

In this section, we present the computational results of the proposed OCGA and the comparisons with two other variants of GA namely Standard Genetic Algorithm (SGA) and MultiCrossover Genetic Algorithm (MXGA) proposed by (Lee et al., 2007). The algorithms are coded in C language and implemented on a Pentium 4, 2.0 GHz computer with 2.0 GB RAM.

We genarated problem instances with 50 and 100 jobs, and with 4, 8 and 12 families. Jobs are distributed uniformly across families, so that each family contains $\lfloor N/F \rfloor$ or $\lceil N/F \rceil$ jobs. In each problem, processing times are randomly generated integers from an uniform distribution defined on [1, 100]. We also generated five sets of integer due dates from the uniform distribution $[0, \alpha P]$, where $\alpha \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$, and P is summation of generated processing times in each problem. Based on (Hariri and Potts, 1997), the three setup times class A, B and C are randomly generated integers from the three uniform distributions [1, 100], [1, 20] and [101, 200] respectively. For each combination of N, F, α and setup times class, five problem instances are created. We used the lower bound proposed by (Hariri and Potts, 1997) to assess the quality of solutions generated by the algorithms. Algorithms are compared by listing, for each combination of value N, F, α and setup times class, the average relative percentage deviation (*ARD*) and the maximum relative percentage deviation (*MRD*) of the heuristic solution value from the lower bound. The *ARD* and *MRD* formulas are shown in equations (2) and (3) respectively.

$$ARD = \frac{\sum_{i=1}^{I} \sum_{r=1}^{R} (\frac{UB_{ir} - LB_i}{LB_i} \times 100\%)}{I \cdot R} \tag{2}$$

$$MRD = \max_{\substack{i=1,2,\ldots,I \\ r=1,2,\ldots,R}} \{\frac{UB_{ir} - LB_i}{LB_i} \times 100\%\} \tag{3}$$

where,

I = number of problem instances with the relevant combination of parameters;

R = number of repeated runs for problem instance i ($i = 1, 2, \ldots, I$);

UB_{ir} = heuristic solution found in rth run of problem instance i;

LB_i = lower bound of the problem instance i.

3.1 Competitors

We use two variants of GA namely SGA and MXGA to compare with our proposed OCGA. In the case of SGA, a standard 1-point crossover operator is applied to produce two offspring from two selected parents, while a reproduction procedure is used when the crossover does not apply to the selected parents. Also the replacement scheme employed in the SGA is the steady-state replacement scheme.

The differences between the MXGA proposed by (Lee et al., 2007) and our OCGA are with regards to the use of the crossover operator, swap operator and the mutation operator. The MXGA selects offspring for the population from a candidate list of temporary offspring generated via F-point crossover operator. During the swap operator, a swap point is randomly selected within a parent and then the substrings separated by the swap point are exchanged to form a new offspring. Two mutation operators are used in MXGA as follows: first, an offspring is selected based on an individual mutation probability, then each element in the selected offspring is visited and altered with a gene mutation probability.

3.2 Results and Discussions

In this computational experiment, we used the problem instances described earlier. For each combination of problem instances, 30 runs were performed. In order to have a fair comparison between different algorithms, we employed a duration of 15 CPU seconds per run in this experiment. Table 1 shows the computational results in which for each algorithm, the entries report the average value of *ARD* and *MRD* computed over the five problem instances with five combinations of due dates (i.e. 750 runs). For each setup class, the final line gives the average over all values of N and F. The final line of Table 1 gives the overall average value over all setup classes.

It is clear from Table 1 that the OCGA performs better than the SGA and MXGA algorithms. This indicates that our proposed algorithm is able to produce better quality solutions compared to others. We have also found that computational difficulty as measured by relative deviation from the lower bound increases with problem size. In the case of setup time class C with large setup time, jobs tend to form a large batch size with more jobs in a batch to reduce the need of setup time between batches from different families. Therefore, more jobs will miss their assigned due dates. However with a small setup time similar to setup time class B, more jobs will meet their respective due dates. Hence when the setup time is small, more batches are formed which means fewer jobs are to be processed per batch.

4. Conclusion

This paper presents a genetic algorithm that uses an optimised crossover operator to solve the problem of $1|s_f|L_{\max}$. Various techniques have also been introduced into the proposed algorithm to further enhance the solutions quality. The computational results indicated that the proposed algorithm is able to generate better quality solutions compared to other variants of genetic algorithms. As for future work, it may be interesting to develop OCGA for other optimality criterion such as minimising the total (weighted) tardiness/earliness.

References

Aggarwal, C. C., Orlin, J. B. & Tai, R.P. (1997). Optimized crossover for the independent set problem. *Operations Research*, 45, 226-234.

Ahuja, R. K., Orlin, J. B. & Tiwari, A. (2000). A greedy genetic algorithm for the quadratic assignment problem. *Computers & Operations Research*, 27, 917-934.

Allahverdi, A., Ng, C. T., Cheng, T. C. E. & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187, 985-1032.

Baker, K. R. (1999). Heuristic procedures for scheduling job families with setups and due dates. *Naval Research Logistic*, 976-991.

Baker, K. R. & Magazine, M. J. (2000). Minimizing maximum lateness with job families. *European Journal of Operational Research*, 127, 126-139.

Balas, E. & Niehaus, W. (1996). Finding large cliques in arbitrary graphs by bipartite matching. In D. S. Johnson & M. A. Trick (Eds.), *Clique, coloring and satisfiability: second DIMACS implementation challenge* (pp. 29-53).

Balas, E. & Niehaus, W. (1998). Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems. *Journal of Heuristics*, 4, 107-122.

Bruno, J. & Downey, P. (1978). Complexity of task sequencing with deadlines, set-up times and changeover costs. *SIAM Journal on Computing*, 7(4), 393-404.

Graham, R. L., Lawler, E. L., Lenstra, J. K. & Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic machine scheduling diseases: a survey. *Annals of Discrete Mathematics*, 5, 287-326.

Hariri, A. M. A & Potts, C. N. (1997). Single machine scheduling with batch setup time to minimize maximum lateness. *Annals of Operations Research*, 70, 75-92.

Holland, J. H. (1975). *Adaptations in natural and artificial systems*. Ann Arbor: The University of Michigan Press.

Jackson, J. R. (1955). *Scheduling a production line to minimize maximum tardiness*. Los Angeles: University of California.

Lee, L. S., Potts, C. N. & Bennell, J. A. (2007). A genetic algorithms for single machine family scheduling problem. *Proceedings of 3rd IMT-GT 2007 regional conference on mathematics, statistics and applications* (pp. 488-493). Malaysia: Penang.

Mason, A. J. (1992). *Genetic algorithms and scheduling problems*, PhD Thesis, University of Cambridge, UK.

Monma, C. L. & Potts, C. N. (1989). On the complexity of scheduling with batch setup time. *Operations Research*, 37(5), 798-804.

Pan, J. C. H., Chen, J. S. & Cheng, H. L. (2001). A Heuristic approach for single machine scheduling with due dates and class setups. *Computers & Operations Research*, 28, 1111-1130.

Potts, C. N. & Kovalyov, M. Y. (2000). Scheduling with batching: a review. *European Journal of Operational Research*, 120, 228-249.

Zdrzałka, S. (1995). Analysis of approximation algorithms for single machine scheduling with delivery times and sequence independent batch setup times. *European Journal of Operational Research*, 80, 371-380.

Table 1. Comparative computational results (15 CPU seconds per run)

Setup Class	N	F	SGA		MXGA		OCGA	
			ARD	MRD	ARD	MRD	ARD	MRD
A	50	4	21.50	91.09	16.86	76.46	15.87	69.25
		8	19.77	62.17	14.63	52.87	13.52	46.36
		12	16.25	45.59	11.00	36.23	10.10	29.07
	100	4	22.85	114.09	19.13	90.31	18.53	84.15
		8	28.58	107.39	21.20	85.88	19.29	76.21
		12	34.81	93.21	19.31	67.02	17.77	58.36
	Average		23.96	85.59	17.02	68.13	**15.85**	**60.57**
B	50	4	7.20	39.51	5.16	30.75	4.70	29.39
		8	9.53	50.39	6.94	35.90	5.16	30.16
		12	8.50	49.06	6.76	44.04	5.37	37.11
	100	4	7.88	38.01	5.87	29.71	5.73	27.26
		8	14.00	89.70	8.21	34.21	7.09	34.05
		12	18.19	78.20	8.12	39.88	6.79	38.12
	Average		10.88	57.48	6.84	35.75	**5.81**	**32.68**
C	50	4	32.47	99.07	20.94	71.77	19.96	66.20
		8	23.93	56.71	15.52	40.75	15.09	36.18
		12	15.99	30.73	11.07	25.74	9.84	20.06
	100	4	43.71	136.61	27.98	92.84	26.92	91.48
		8	39.65	96.14	26.63	79.12	24.97	70.43
		12	49.42	84.32	22.91	60.75	20.88	54.55
	Average		34.20	83.93	20.84	61.83	**19.61**	**56.48**
AVERAGE			23.01	75.67	14.90	55.24	**13.76**	**49.91**