Turkish Online Journal of Qualitative Inquiry (TOJQI)
Volume 12, Issue 7, July, 2021:1539 – 1547

Minimizing makespan in a single batch machine using genetic algorithm

Ho Jin Kai[a], Yasothei Suppiah[b], Ajitha Angusamy[c], Goh Wei Wei[d], Noradzilah Ismail[e]

[a,b,d,e] Faculty of Engineering & Technology, Multimedia University, Malaysia
[c] Faculty of Business, Multimedia University, Malaysia

Abstract

This paper deals with a single batch machine scheduling problem to minimize makespan with the consideration of sequence dependent setup time. As the problem is NP hard, a genetic algorithm is developedto provide a solution to this problem. Furthermore, dispatching heuristics such as shortest processing time (SPT), longest processing time (LPT), smallest job size (SJS) and largest job size (LJS) are developed to provide initial solutions to the genetic algorithm. All the developed algorithms and simulation are done using Spyder (Python 3.8) software. Computational results show that the genetic algorithm has outperformed the dispatching heuristics in minimizing the makespan.

Keywords: makespan, scheduling, genetic algorithm, dispatching heuristics

1. Introduction

Scheduling is a process of making decisions in a manufacturing environment with various restricted constraints with the aim of achieving goals in an industry. Decisions made from solving the scheduling problem are instrumental in accomplishing the tasks and goal of the industry. In recent years, the complexity in making these decisions have increased tremendously due to very high production volumes, higher product portfolio based on customers' needs and higher pressure by the management to save on production and energy costs[8]. Besides that, efficient algorithms are the sought after tools to cope with these uncertainties in production.

Batch scheduling problems commonly appears in industrial processes, manufacturing and cellular assembly systems [1]. Parallel batching is categorised as processing several jobs simultaneously on a processor in which the processing time of a batch is the largest processing time of jobs in [5]. The parallel batching machine has the characteristic of processing a group of jobs with the condition that the sum of job sizes in the batch should not exceed the capacity of the machine. One of the most widely industrial application of parallel batch machine is in electronic assemblies and burn in oven [6]. Since the batch parallel machines often leads to a bottleneck situation, therefore efficient scheduling is of a great concern to maximize throughput [5] and [7].

A constraint programming that exploits a new optimization constraint to minimize maximal lateness for a batch processing machine is presented in [10]. However, this method provides optimal solutions for small size job instances only. Furthermore, developing exact algorithms for industrial sized problems remains a challenging issue till date [11].

A genetic algorithm for jobs scheduling of non-similar sizes on a single-batch-processing system is used in [2] to minimize makespan. They suggested two separate genetic algorithms based on various encoding schemes, the first is a sequence-based GA, which uses GA operators to generate random work sequences and then grouping the jobs. The second is a hybrid batch-based GA, which creates batches of jobs randomly using crossover and mutation operations and ensures feasibility by relying on problem information through a heuristic method. This batch-based hybrid GA is combined with a local search heuristic based on the problem characteristics to direct the search to near optimal or optimal schedules.A hybrid genetic local search algorithm was proposed in [3] for the unrelated parallel system scheduling problem with the aim of minimizing the maximum completion time. They suggested a chromosome structure made up of arbitrary key numbers which leads to very effective computational results in achieving the objective of the scheduling.

A genetic algorithm for the unrelated parallel machine scheduling problem [4], takes into account machine and job sequence based setup times. A quick local search and a local search enhanced crossover operator are included in the proposed genetic algorithm. After some calibrations, two versions of the algorithm are obtained and are its performance are compared to the most widely used approaches in the literature. To carry out the statistical experiments, they also establish a benchmark of small and large instances and conclude that the proposed approach outperforms the other evaluated methods in a benchmark collection of instances after conducting a numerical and statistical study.A single batch processing machine to minimize makespan is considered in [9] where the machine's capacity is represented by a two-dimensional rectangle and a job occupies a rectangle of its dimension. They developed a biased random-key genetic algorithm together with hybrid bin load algorithm. These algorithms have shown comparable results to the mixed integer programming solver, CPLEX for moderate sized instances. On the other hand, the developed algorithms have outperformed CPLEX on large size instances with a reasonable computational time.

This paper deals with a single batch machine scheduling problem to minimize makespan. The scheduling problem in this paper is defined as following. There are N jobs which are categorised into F families according to their job characteristics. Jobs belonging to same families will be grouped into batches where the total jobs sizes in a batch does not exceed the capacity of the machine. Once a batch starts itsprocessing, it cannot be interrupted and no jobs can be removed from or added into the batchits processing is completed. The processing time (p_b) of a batch is equivalent to the longest processing time of a job in that particular batch. A set-up time (t_{ij}) occurs whenever there is a switch in processing of batches from one family to a batch originating from another family in the scheduling sequence. The makespan for the sequence of batches are calculated by addingup the processing time of the batches and the setup times.

2. Methodology

In this paper, we have developed four dispatching heuristics which are SPT, LPT, SJS, LJS and a metaheuristic, genetic algorithm to solve the scheduling problem. All data simulation and the heuristics are developed using Spyder (Python 3.8).

Development of Dispatch Heuristics

- SPT:Jobs are sequenced as according to the increasing order of their processing time.
- LPT: Jobs are sequenced as according to the decreasing order of their processing time.
- SJS: Jobs are sequenced as according to the increasing order of their job size.
- LJS: Jobs are sequenced as according to the decreasing order of their job size.

For every dispatch heuristic, once the jobs are sequenced according to their priority rules, jobs from the same family are grouped into batches. The total job size of the batch cannot exceed the capacity of the machine. The total completion time of the final schedule of batchesare calculated for each of the dispatch heuristic takes into consideration of the processing time and the sequence dependent setup time whenever there is a change of batches from one family to another in their respective sequence.

Development of Genetic Algorithm

Genetic Algorithm adopts the concept of biological evaluation theory whereby offspringchromosomes are created at every iteration from parent chromosomes. Furthermore, the genetic algorithm also fosters the principal of survival of the fitness whereby chromosomes that contributes the least to the objective function will be left to die. As the genetic algorithm requires initial population to start off its algorithm, it has been a common approach to generate initial population either from random instances or from other heuristics. Here, we have adopted four dispatch heuristics which are SPT, LPT, SJS and LJS to generate initial population for the genetic algorithm. As the population size of the genetic algorithm has been fixed before starting the genetic algorithm, only the chromosome with good fitness will be selected to reproduce new offspring in every iteration. Each job can be viewed as a gene in a chromosome and each sequence of scheduled jobs can be viewed as a chromosome. Every chromosome has its own fitness function as it is measured by the objective function of the problem which is the total completion time in our case. The fitness function helps in determining the quality of the chromosome. At every iteration, once the parents are chosen from the population, new chromosomes (children)are born after applying operators such as crossover and mutation to the parent chromosome. In this paper, a position based crossover genetic operator is used. Based on the random number generated, a set of genes that equals to the random number is copied from the first parent to the child chromosome. The rest of the genes which are not in the child chromosome yet, follow the sequence that appear in parent 2. Another child is created by using the mutation operator by swapping the position of two random genes of the child chromosome from the crossover operation.As every new chromosome are created, they are evaluated on based on their fitness function. They too become the part of the growing population. If the population of chromosomes exceeds the number of maximum populations, then the chromosomes with the least fitness will be left to die in order to make room for the new chromosomes that are stronger.The genetic algorithm stops when the stopping criteria is met usually when a maximum number of iterations is reached.

Step 1: Initial population of chromosomes is generated using the outcomes of the job sequences and the makespan values of the dispatching heuristics SPT, LPT, SJS and LJS. Set the maximum number of population and maximum number of iterations (termination condition).

Step 2: Parents chromosomes are chosen by selecting any 2 chromosomes from population.

Step 3:Position based crossover genetic operators are applied to generate a child. The front part of the genes in parent 1 iscopied to the front part of the child chromosome. The number of genes chosen to be copied is based on the random number generated which is less than the number of jobs in the parent chromosome. The rest of the genes in child chromosome are arranged according to the sequence of these genes that appear in parent 2.

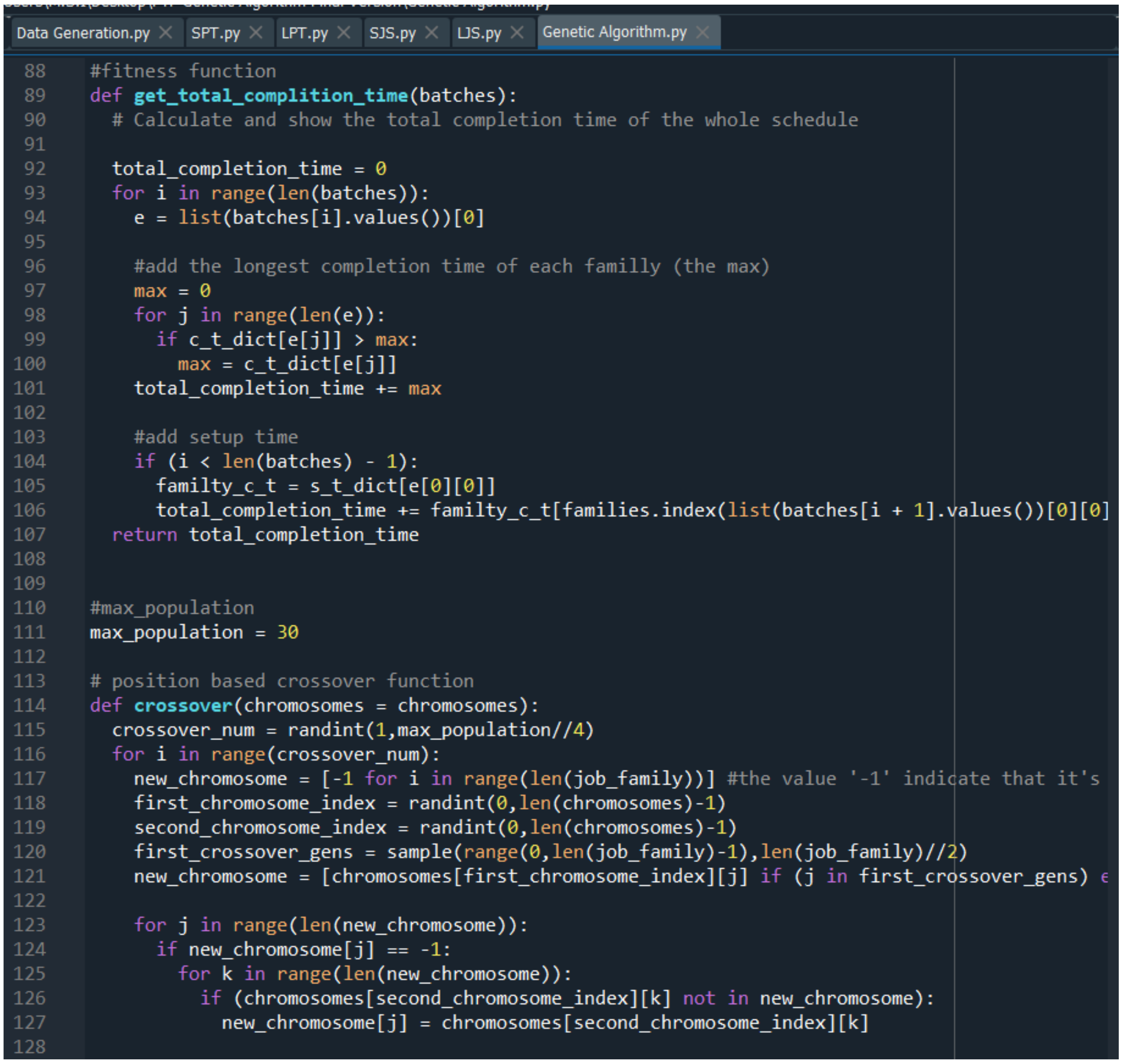
Step 4:Another child chromosome is generated by applying mutation genetic operators. Two genes are chosen randomly and their positions are swapped in the child chromosome from step 3

Step 4:For every child chromosome generated from steps 3 and 4, group the jobs from the same family into batches and the total job size of a batch cannot exceed the capacity of the machine.

Step 5:Evaluate the makespan(fitness value) of all the children chromosomes and insert them into the population if the population size has not reach maximum number of population.

Step 6: If the population of chromosomes exceeds the number of maximum populations, then delete the chromosome with the least fitness to make room for the new chromosomes.

Step 7: If the maximum number of iterations is satisfied then stop and return the best chromosome with the list of jobs in every batch and its makespan value; otherwise, go to Step 2.



```python
#fitness function
def get_total_complition_time(batches):
  # Calculate and show the total completion time of the whole schedule

  total_completion_time = 0
  for i in range(len(batches)):
    e = list(batches[i].values())[0]

    #add the longest completion time of each familly (the max)
    max = 0
    for j in range(len(e)):
      if c_t_dict[e[j]] > max:
        max = c_t_dict[e[j]]
    total_completion_time += max

    #add setup time
    if (i < len(batches) - 1):
      familty_c_t = s_t_dict[e[0][0]]
      total_completion_time += familty_c_t[families.index(list(batches[i + 1].values())[0][0]
  return total_completion_time


#max_population
max_population = 30

# position based crossover function
def crossover(chromosomes = chromosomes):
  crossover_num = randint(1,max_population//4)
  for i in range(crossover_num):
    new_chromosome = [-1 for i in range(len(job_family))] #the value '-1' indicate that it's
    first_chromosome_index = randint(0,len(chromosomes)-1)
    second_chromosome_index = randint(0,len(chromosomes)-1)
    first_crossover_gens = sample(range(0,len(job_family)-1),len(job_family)//2)
    new_chromosome = [chromosomes[first_chromosome_index][j] if (j in first_crossover_gens) e

    for j in range(len(new_chromosome)):
      if new_chromosome[j] == -1:
        for k in range(len(new_chromosome)):
          if (chromosomes[second_chromosome_index][k] not in new_chromosome):
            new_chromosome[j] = chromosomes[second_chromosome_index][k]

```

Figure 2: Screenshot of python program for the genetic algorithm

3. Results and Discussion

Random data which are extracted from literature [1] is generated for the job parameters to test the effectiveness of the genetic algorithm performance in minimizing the makespan of the scheduling problem presented in this paper. All the random instances, developed heuristics and the simulation experiments are run using the Spyder (Python 3.8). The data for 25 jobs and the parameters used are shown in Table 1.

Table 1: Random instances for 25 jobs

Parameters	Values
Number of Families of Jobs	5
Number of Jobs in Each Family	5
Lower and Upper Boundaries of Job Size	[20, 50]
Lower and Upper Boundaries of Processing Time of Jobs	[20, 50]
Lower and Upper Boundaries of Set Up Time for Different Families of Jobs	[20, 50]
Machine Capacity	100

Based on Table 1, 5 sets of instances have been generated. The makespan values are recorded for each of the heuristics. As for the genetic algorithm, there are 5 types of genetic algorithm was tested in terms of the maximum iteration for the stopping criteria. GA(1000) is where the stopping criteria of the algorithm is set to 1000 iterations, The stopping criteria for GA(2000), GA(3000) ,GA(4000) and GA(5000) is set at 2000 iterations, 3000 iterations, 4000 iterations and 5000 iterations respectively. For the dispatch heuristics SPT, LPT, SJS and LJS, each of the 5 instances are run once and the total completion time is recorded. However, for each type of the genetic algorithm, each instances are run 5 times and the average of the makespan is recorded. This is because the genetic algorithm has features of crossover and mutation which results in different combinations of chromosomes at every iteration. The value of the makespan can be observed at Table 2 for all the 5 sets of instances.

Table 2: Makespan values for heuristics

	SPT	LPT	SJS	LJS	GA(1000)	GA(2000)	GA(3000)	GA(4000)	GA(5000)
Instance 1	1056	1097	1211	1266	800	792	774	702	704
Instance 2	1202	1209	1445	1273	755	839	755	741	816
Instance 3	1296	1415	1419	1625	890	897	824	853	568
Instance 4	1545	1432	1431	1473	820	756	856	798	755
Instance 5	1389	1606	1687	1659	786	754	783	692	792
Average	1297.6	1351.8	1438.6	1459.2	810.2	807.6	798.4	757.2	727

It can be seen from Table 2, the genetic algorithm produces better results for the makespan values for all the 5 instances. The last row at Table 2 shows the average performance of each of the heuristics. Among the dispatching heuristics, SPT tends to show the best performance and the performance of the genetic algorithm improves as more iterations are used as the stopping criteria. The average of the 5 instances are shown clearly in Figure 3.

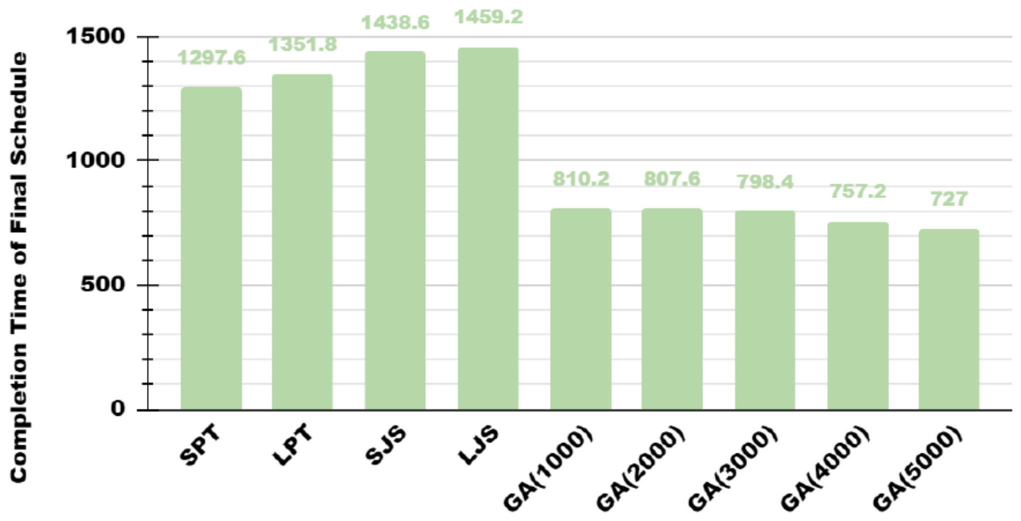


Figure 3: Average of the completion time (makespan value) for 25 jobs

Since each type of genetic algorithm is run for 5 times for each instances, the best makespan value (minimum value) from the 5 runs are recorded for each instances at Table 3. Hence, the last row provides the average of the 5 values recorded for each type of the genetic algorithm.

Table 3: Best objective function value for each type of GA for each instances

Average of the best makespan value in 5 runs	GA(1000)	GA(2000)	GA(3000)	GA(4000)	GA(5000)
Instance 1	795	776	752	690	694
Instance 2	745	825	743	728	799
Instance 3	876	884	803	843	554
Instance 4	806	738	835	785	743
Instance 5	774	732	769	692	779
Average	799.2	791	780.4	747.6	713.8

In order to study how much of improvement made by the genetic algorithm in terms of the total completion time with respect to each of the dispatching heuristics, we have calculated the percentage of improvement (I) by using the formula below:

$$I = \frac{\dfrac{\sum_{n=1}^{5} C_{T_{dhn}}}{5} - \dfrac{\sum_{n=1}^{5} C_{T_{GAn}}}{5}}{\dfrac{\sum_{n=1}^{5} C_{T_{dhn}}}{5}} \times 100\%$$

$C_{T_{dhn}}$ = makespan value of dispatching heuristic for instance n

$C_{T_{GAn}}$ = average makespan value of genetic algorithm for instance n

Table 4 provides the percentage of improvement of each type of the genetic algorithm from each of the dispatch heuristics. In general, the genetic algorithm has improved the performance of the dispatching heuristics in terms of the makespan values in between the range of 37.56% to 50.18%.

Table 4: Percentage of improvement of each type of genetic algorithm

Percentage of improvement of each type of GA compared to dispatching heuristics	Dispatching heuristics			
	SPT	LPT	SJS	LJS
GA(1000)	37.56	40.07	43.68	44.48
GA(2000)	37.76	40.26	43.86	44.65
GA(3000)	38.47	40.94	44.50	45.29
GA(4000)	41.65	43.99	47.37	48.11
GA(5000)	43.97	46.22	49.46	50.18

In terms of the computational time taken to produce the output for each of the heuristics, the dispatching heuristics are the fastest. Therefore, we have recorded the computational time for the genetic algorithm as it has more complex features in its algorithm and various stopping criteria. Figure 4 provides the average computational time taken by each type of the genetic algorithm. It can be seen that as the number of iterations increases the time taken to produce results also increases.

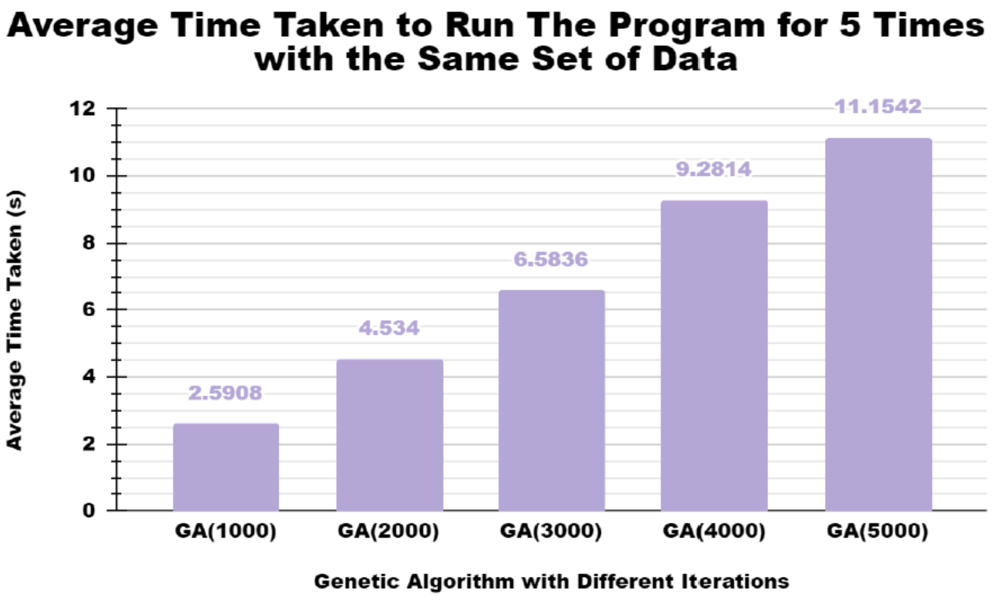


Figure 4: Average computational time by each of the genetic algorithm

Similar experiments are carried for 50 jobs in which all parameters used for jobs and machines are the same as in Table 1 except that there are now 10 jobs in each family.

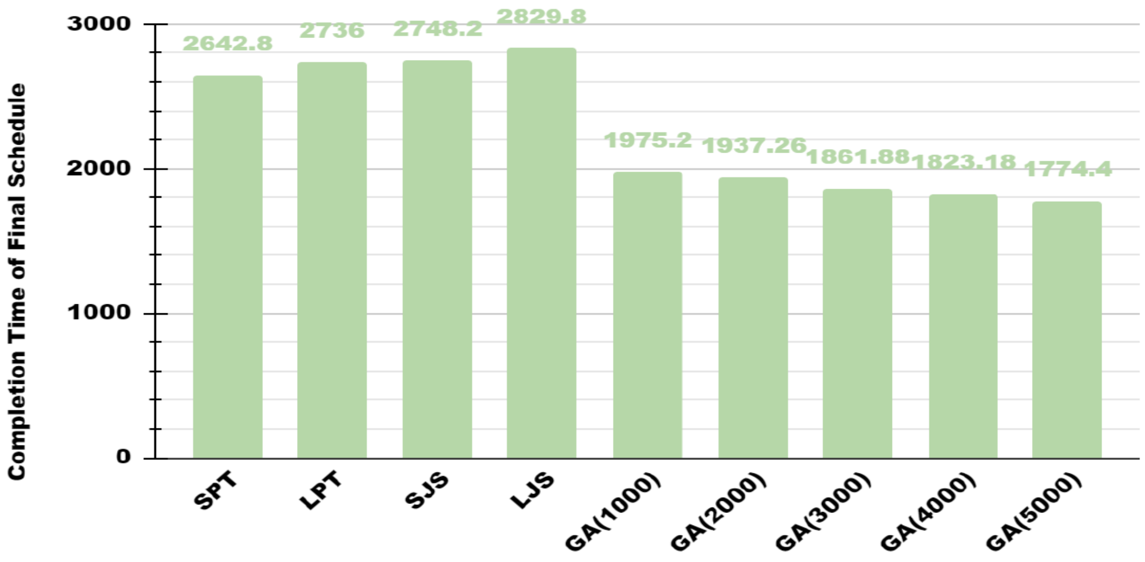


Figure 5: Average of the completion time (makespan value) for 50 jobs

Figure 5 provides the average makespan value out of the 5 instances for each of the heuristics for the case of 50 jobs. SPT shows the best results among all the dispatching heuristics whereas LJS shows the worst results. All types of genetic algorithm show improvement to the dispatching heuristics and the performance of the genetic algorithm improves as more iterations are being used in its structure.

Similar to the earlier explanation on Table 3, each type of genetic algorithm is run for 5 times for each instances, the best makespan value (minimum value) from the 5 runs are recorded for each instances at Table 5. Hence, the last row provides the average of the 5 values recorded for each type of the genetic algorithm.

Table 5: Best objective function value for each type of GA for each instances

Average of the best makespan value in 5 runs	GA(1000)	GA(2000)	GA(3000)	GA(4000)	GA(5000)
Instance 1	1884	1907	1635	1328	1295
Instance 2	1819	1674	1738	1650	1569
Instance 3	1891	1866	1839	1737	1657
Instance 4	1836	1817	1793	1752	1684
Instance 5	1894	1879	1859	1755	1680
Average	1864.8	1828.6	1772.8	1644.4	1577

Futhermore, the genetic algorithm have improved the results compared to the dispatching heuristics in the range of 25.26% to 37.30% for the case of 50 jobs. However the average computational time (in seconds) taken to produce the final results are 7.05s, 14.06s, 20.33s, 27.36s and 33,69s for GA(1000), GA(2000), GA(3000), GA(4000) and GA(5000) respectively. The time taken are longer compared to the case of 25 jobs as more jobs require more computational time to produce the solution.

4. Conclusion

This paper provides a solution for the single batch machine scheduling problem with the consideration of sequence dependent setup time. The objective of this paper is to provide a good schedule that minimises the makespan for a single batch machine scheduling problem. Since the problem in NP hard, a metaheuristic genetic algorithm is developed in this paper to find a near optimal solution. There are four dispatching heuristics developed in this paper to be used as an initial population for the genetic algorithm. The result obtained from the genetic algorithm has been compared with the results from the dispatching heuristic methods in terms of the makespan value. Based from the experiments outcome, the genetic algorithm is able to produce a good improvement within a reasonable time compared with the dispatching heuristics. This study could be extended to other scheduling environments such as parallel machines, flow shops or jobs shops as a future direction.

References

[1] TsiuShuang Chen, Lei Long and Richard Y.K. Fung, “A Genetic Algorithm for the Batch Scheduling with Sequence-Dependent Setup Times”, 2006. Available: https://sci-hub.st/https://link.springer.com/chapter/10.1007/978-3-540-37258-5_147

[2] A. H. Kashan, B. Karimi and F. Jolai, “Effective hybrid genetic algorithm for minimizing makespan on a single-batch-processing machine with non-identical job sizes”, 2005. Available: https://sci-hub.st/https://www.tandfonline.com/doi/abs/10.1080/00207540500525254

[3] Duygu Yilmaz Eroglu, H. Cenk Ozmutlu and Seda Ozmutlu, “Genetic algorithm with local search for the unrelated parallel machine scheduling problem with sequence-dependent set-up times”, 2014. Available: https://sci-hub.st/https://www.tandfonline.com/doi/abs/10.1080/00207543.2014.920966

[4] Eva Vallada and Rubén Ruiz, “A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times”, 2011. Available: https://sci-hub.st/https://www.sciencedirect.com/science/article/pii/S0377221711000142

[5] Rui, X., Huaping, Ch., and Xueping, Li. (2012). Makespan minimization on single batch-processing machine via ant colony optimization. Computers & Operations Research, 39(3), 582-593.

[6] Dauzère-Pérès, S., and Mönch, L. (2013). Scheduling jobson a single batch processing machine withincompatible job families and weighted number oftardy jobs objective. Computers& OperationsResearch, 40(5), 1224-1233

[7] Rocholl, J., Mönch, L. & Fowler, J. Bi-criteria parallel batch machine scheduling to minimize total weighted tardiness and electricity cost. J Bus Econ90, 1345–1381 (2020). https://doi.org/10.1007/s11573-020-00970-6

[8] .Lennart Merkert and Iiro Harjunkoski and Alf Isaksson and Simo Säynevirta and Antti Saarela and Guido Sand.Scheduling and energy – Industrial challenges and opportunities.Computers & Chemical Engineering,72,183-198, 2015.

[9] Xueping Li, Kaike Zhang. Single batch processing machine scheduling with two-dimensional binpacking constraints.International Journal of Production Economics,196, 113-121, 2018.

[10] . Arnaud Malapert, Christelle Guéret, Louis-Martin Rousseau.A constraint programming approach for a batch processing problem with non-identical job sizes. European Journal of Operational Research, 221, 533-545, 2012.

[11] Shaoxiang Zheng and Naiming Xie and Qiao Wu. Single batch machine scheduling with dual setup times for autoclave molding manufacturing.Computers & Operations Research,133,105381, 2021.