A Parallel Adaptive Genetic Algorithm for Job Shop Scheduling Problem

Wathiq N. Abdullah¹ and Salwa A. Alagha²

¹ Department of Computer Science, College of Education for Pure Science/Ibn-Al Haitham/University of Baghdad, Iraq
² Department of Electrical Engineering, University of Technology, Iraq

E-mail: wathiq.n.aa@ihcoedu.uobaghdad.edu.iq

Abstract. In order to enhance the production efficiency, scheduling problem of job-shop has used that thought of complex problem with complicated constraints and structure. This problem is characterized as NP-hard. In most cases, the excessive complexity of the problem makes it difficult to discover the best solution within affordable time. Hence, searching for estimated solutions in polynomial time rather than precise solutions at excessive cost is favored for challenging situations of the problem. In this paper, a parallel genetic algorithm with proposed adaptive genetic operators and migration operation is applied for job-shop scheduling problem. Through tests on numerous different experimental cases, the adaptive operator of genetic algorithm and the parallelism strategy are considerably improving the results effectively while decreasing the computation time. Also, the migration operation gives a greater effect on the performance of the algorithms.

Keywords: Job-Shop, Scheduling, Genetic Algorithm, Adaptive Genetic Operators, Migration Operation, Parallel Algorithm

1. Introduction

Job-Shop Scheduling Problem is a conventional combinational optimization problem. It is common in present day production industries in which numerous processing machines need to be allocated fairly and operation orders of the jobs order need to be sorted well to enhance manufacturing efficiency. Its most important characteristic includes numerous machines with same function but distinct operation time [1]. Efficient methods of solving scheduling problems can have major effects on profitability product quality. The big matter is the minimization of the whole elapsed time between the start of the first operation and the completion of the last operation (the makespan). The shortest makespan is the simplest and most extensively used criterion among any other measures of scheduling quality. In general, the difficulty of the Job-Shop Scheduling problem makes it very difficult for conventional search-based approaches to find near-optima in reasonable time. This has led to recent interest to adopt genetic algorithms to deal with these problems [2]. Genetic algorithm (GA) is a flexible in dealing with complicated problems and normal attribute of parallel processing, that randomly and successfully pattern and search at big state space, and rapidly converge to the optimal or near-optimal solution, so it gets extensively interest in the discipline of scheduling. Experimental tests showed GAs’ scheduling performances in a several different criterions are better than heuristic and stochastic hill-climbing approaches. All these illustrate that GAs are so appropriate to solve scheduling problem [3]. The remainder of this paper is organized as follows: the job-shop scheduling problem, the design of the...
system of solving job-shop scheduling problem using parallel genetic algorithm, and finally, the results of the system and some concluding remarks are given.

2. Job-Shop Scheduling Problem Description
Scheduling for the adaptable activity shop is essential in the two fields of creation administration and combinatorial improvement. Be that as it may, it is very hard to accomplish an ideal answer to this issue in medium and genuine size issue with conventional improvement approaches inferable from the high computational unpredictability [4].

2.1. Problem Description
The JSSP is a scheduling problem that assumes M various machines and N various jobs. Each job consists of Q operations and each operation needs a different machine. The jobs' operations are handled in a fixed processing order which specifies the precedence restrictions. The job's operations are completely ordered so that no job's operation can begin earlier than the completion of its predecessor [5]. The operation sequence of every job is predetermined whereas each operation will be processed on any a machine from a group of candidate machines [6]. To achieve optimal performance, the system should satisfy the scheduling goals, which include: selection of the most appropriate machine for every process, and approve the optimal sequence of processing and time of every work process on every machine [3].

2.2. Problem Formulation
The JSSP is defined by:

\[ \text{J} = \{J_1, J_2, \ldots, J_n\} \] the set of n independent jobs,
\[ \text{O} = \{(O_{11}, O_{12}), (O_{21}, O_{22}), \ldots\} \] the set of operations, where \( O_{ij} \) is operation \( i \) of job \( j \).
\[ \text{M} = \{m_1, m_2, \ldots, m_k\} \] a set of machines.

The goal is to find a schedule of operations that minimizes the completion times of all jobs (MakeSpan of the schedule):

\[ \text{Makespan} = \text{Min} \left( \text{Max} \left( T_1, T_2, \ldots, T_n \right) \right) \] \( ............ (1) \)

where, \( T_j \) is the completion time of job \( J \).

2.3. Parallel Genetic Algorithm for JSSP
The following sections illustrate the application of parallel GA for solving JSSP:

2.4. Parallel GA
GA is a powerful approach used in many various areas to search for a near-optimal solution when searching for the optimal solution is too expensive[7]. Though, parallelization is also an appropriate way to enhance the process time and the efficiency in the exploration of the search space. Certainly, GAs are ‘naturally parallelizable’, for example, their population-based features enable us to assess in a parallel manner the fitness of each individual [7]. To exploit parallelism in GAs, various ways exist; these include: master–slave models, fine-grained models, island models, and hybrid models. Island models are the foremost known to consider on parallel GAs. Populations on islands are free to converge toward totally different sub-optima with a quicker improvement of the average fitness and a migration operator will facilitate combine good features that emerge from different local islands [8]. The advantages of the parallel design are as follows:
• Since more than GA work together, GA get new search points in exploring space via various
techniques, this strategy increases the searching process range and decreases the probability of
premature convergence.
• Because of the independent evolution, individuals from heterogeneous islands gain different
characters from a range of different solutions. Thus, the migration performance is improved [9].
In this paper, three Genetic Algorithms are used in parallel mode. Each GA consists of two main keys:
choosing an encoding representation of the problem; and defining the genetic operators [7].

2.5. Encoding and Fitness Evaluation for JSSP

• Encoding Representation of a schedule
As a schedule consists of a sequence of total operations $OP$, it can be represented as follows:

$$OP = \{ op_{ij} \mid i = 1, 2, \ldots, N \text{ and } j = 1, 2, \ldots, M \}$$

Where: $op$ is the operation, $i$ is the job number, $j$ indicates the operation number of job $i$, $N$ is the set of
jobs, and $M$ is the set of available machines. The $ij$ indices of the operation $op$ are encoded as a
sequence number $(index)$ of a chromosome's gene which can be represented by two fields: the first
field consists of the machine's number and the second field of a gene is the completion time of the
operation [2].
The chromosome length is equal to the product of the number of jobs and the total number of
machines in a given problem [10].

• Evaluation of Fitness Function
The fitness value can be evaluated by summing the total completion time of the operations on each
machine (TCT). The total completion time with maximum value is the chromosome's fitness value [2],
as follows:

$$\text{Fitness Value} = \text{Maximum of (TCTx)} = \text{Max (TCTx)} \quad \ldots (3)$$

Where:
\begin{align*}
x & : \text{is the machine number, } 1 \leq x \leq M, \\
M & : \text{is the no. of machines.}
\end{align*}

2.6. The Operators and the Proposed Adaptive Operators of GA
The three genetic algorithms adopted in this paper use several types of GA operators. Some of them
are the same as original GA operators and the other are improved and adapted to give best results.

• Original GA Operators
For selection operator, the system uses tournament selection technique with tournament of two and
three. For crossover operator, one-point crossover and two-point crossover operator are used. For
mutation operator, random selection of chromosome and randomly replace a position in it with
random value [11].

• The Proposed Adaptive Operators of GA
i. Adaptive Selection Operator:
Because of the size of the population of every generation is constant, each time one
individual chosen, there is an excessive possibility that individuals with high fitness are selected over
and over whereas individuals with low fitness aren't selected at all. However, when the generation number increased, the population will contain many good but identical individuals.

The proposed selection operator is as follows: in each tournament selection, the individual having the greater fitness is added to the new population and removed from the parent population. Such that, every individual can be chosen solely as soon as such that the population's diversity is ensured.

ii. Adaptive Crossover Operator
In this paper, a special crossover operator is adopted which randomly selects two individuals, and then checks whether they have common points (i.e., common machine number within a schedule). If not, then the crossover will not be performed and the two parents from the population are copied to the population of the next generation. If they have only one common point, then that point is the crossover point. If there are more than one common points, then the crossover point is randomly selected.

iii. Adaptive Mutation Operator
The mutation operator is improved as follows: after a random selection of an operation within a schedule, this operation is reassigned to a machine away from the original machine in two locations in circular style. That is, if the machine number is x, then reassign to machine number x+2, and if the original machine is the last machine then reassign to machine 2.

2.7. The Migration Operation
The migration operation is the process of swapping individuals amongst the subpopulations of the GA. The operation of migration improves the population's genes and consequently will increase the diversity and speed up the convergence of the algorithm.

The essential migration's parameters are the interval of migration, rate of migration, the technique for choosing migrating individuals, the technique for changing an individual by the new one in the receiving subpopulation, and the type of topology.

The migration topology adopted in this paper is based on the unidirectional ring topology by which the communications between islands are done on a unidirectional ring, and hence an island can send migrants to its next neighbors and receive migrants only from its previous neighbors. Each sub-population sends and receives the best individual from the neighbor sub-population, and exchanges the worst one in its pool by the received individual. The rate of migration was determined as a fixed number of the best individuals in the population in each interval. The interval in terms of generations in GA [12].

3. The Implementation of The System
The system uses three genetic algorithms (GA1, GA2, and AGA which stands for Adaptive Genetic Algorithm) work in parallel mode. It starts by dividing the initial population into three parts. GA1 is applied on the first part, GA2 is applied on the second part, and AGA is applied on the third part.

The types of genetic operators used in each genetic algorithm are shown in Table (1).

| GA Operators | GA1                  | GA2                  | AGA                  |
|--------------|----------------------|----------------------|----------------------|
| Selection    | Tournament (Size= 2) | Tournament (Size= 3) | Adaptive selection   |
| Crossover    | One-point            | Two-point            | Adaptive Crossover   |
| Mutation     | Classical (random change) | Classical (random change) | Adaptive Mutation   |
The detailed steps of the system are shown in Figure (1).

![Figure (1): The Steps of The System of The Parallel GA.](image)

4. **Experimental Settings and Results**

Various cases of different job-shop problem are taken. However, All the three genetic algorithms use the same case instances in order to compare the implementation results among them.

Firstly, Table (2) shows various problem instances. These instances are different in the number of machines in the system and the number of jobs.
Table (2): The Problem Instances Used by The System

| Instance number | No. of Machines | No. of jobs |
|-----------------|-----------------|-------------|
| 1               | 6               | 6           |
| 2               | 10              | 10          |
| 3               | 8               | 8           |
| 4               | 10              | 4           |
| 5               | 10              | 5           |

The system specifications and the parameters used with their assigned values are described in Table (3).

Table (3): The System’s Parameters Settings

| Parameter Name                    | Value            |
|-----------------------------------|------------------|
| Sub-population size               | 50 individuals   |
| Max No. of Generations            | 500              |
| Crossover Rate (pc)               | 0.7              |
| Mutation Rate (pm)                | 0.1              |
| No. of Migrant Individuals        | 4                |

The adopted migration policy is done by the selection of the best individuals from the sending island and the replacement of the worst individuals in the destination island.

The result of applying each genetic algorithm on any of the problem instances is a schedule having a minimum makespan of a schedule. The Table (4) and Figure (2) show the makespans from GA1, GA2, and AGA.

Table (4): The resultant makespan from GA1, GA2, and AGA

| Instance Number | Makespan of GA1 | Makespan of GA2 | Makespan of AGA |
|-----------------|-----------------|-----------------|-----------------|
|                 | Without Migration | With Migration | Without Migration | With Migration | Without Migration | With Migration |
| 1               | 68              | 54              | 70              | 50              | 62              | 48              |
| 2               | 196             | 112             | 185             | 88              | 86              | 75              |
| 3               | 107             | 89              | 101             | 84              | 84              | 61              |
| 4               | 240             | 205             | 232             | 194             | 190             | 172             |
| 5               | 181             | 167             | 177             | 155             | 154             | 128             |

From Table (4) and Figure (2), it is obvious that the best makespan is from the schedule resulted from AGA with migration operation, while the worst makespan resulted from GA1.
The converged generation numbers for GA1, GA2, and AGA at which the algorithms reach the best schedule are shown in Table (5). Table (5) and Figure (3) show the converged generation number with and without applying the migration operation.

Table (5): The Converged Generation Number from GA1, GA2, and AGA

| Instance Number | Gen. No. of GA1 | Gen. No. of GA2 | Gen. No. of AGA |
|-----------------|----------------|----------------|----------------|
|                 | Without Migration | With Migration | Without Migration | With Migration | Without Migration | With Migration |
| 1               | 165             | 144            | 146            | 142            | 90              | 77             |
| 2               | 254             | 219            | 202            | 197            | 132             | 101            |
| 3               | 137             | 130            | 129            | 111            | 107             | 85             |
| 4               | 277             | 215            | 238            | 221            | 134             | 120            |
| 5               | 116             | 103            | 108            | 106            | 98              | 95             |

The results displayed in Table (4) and Table (5) showed that the GA2 gives better makespan and better converged generation number than the GA1. This is due to: (i) the larger tournament size of selection operator adopted in GA2 than that in GA1, (ii) the two-point crossover operator used in GA2 that provides more diversity than one-point crossover in GA1.

Figure (2): The makespans from GA1, GA2, and AGA.
5. Conclusions

In this paper, we present a system of implementation of a parallel genetic algorithm to find the best schedule of a job-shop scheduling problem. The system proposes adaptations for the three GA operators. It demonstrates that by dividing the initial population and uses adaptive GA operators in parallel mode give best schedule, and less computation time.

The system uses three genetic algorithms each with different type of genetic operators. The use of adaptive selection operator, adaptive crossover operator, and adaptive mutation operator in the third algorithm has significantly improves the system performance in terms of the resulted schedule and time it takes to find the best solution. The application of migration operation among the three genetic algorithms greatly enhanced the system performance.
References:

[1] Zhang, M., et al., *A competitive and cooperative Migrating Birds Optimization algorithm for vary-sized batch splitting scheduling problem of flexible Job-Shop with setup time*. Simulation Modelling Practice and Theory, 2020. 100: p. 102065.

[2] Abdullah, W.N., *Solving Job-Shop Scheduling Problem Using Genetic Algorithm Approach*. 2011(70): p. 241-253.

[3] Hu Li, Z.L., Ruiduan Yang, Huawei Lu, Youchao Zhang, *A Flexible Job-shop Scheduling for Small Batch Customizing*. 2016: p. 2179-2201.

[4] *Solving Flexible Job Shop Scheduling Problem Using Meerkat Clan Algorithm*. Iraqi Journal of Science, 2018. 59(2A).

[5] Abdullah, W.N., *Solving Job-Shop Scheduling Problem Using a Developed Particle Swarm Optimization Algorithm*. International Journal of Science and Research (IJSR), 2018. 7(1): p. 1845-1848.

[6] Huizhi Ren, H.X., Shenshen Sun, *Immune Genetic Algorithm for Multi-objective Flexible Job-shop Scheduling Problem*. 28th Chinese Control and Decision Conference (CCDC), 2020: p. 2167-2171.

[7] Salza, P. and F. Ferrucci, *Speed up genetic algorithms in the cloud using software containers*. Future Generation Computer Systems, 2019. 92: p. 276-289.

[8] Luo, J., et al., *GPU based parallel genetic algorithm for solving an energy efficient dynamic flexible flow shop scheduling problem*. Journal of Parallel and Distributed Computing, 2019. 133: p. 244-257.

[9] Luo, J., et al., *Solving the dynamic energy aware job shop scheduling problem with the heterogeneous parallel genetic algorithm*. Future Generation Computer Systems, 2020. 108: p. 119-134.

[10] Kaouther Ben Ali, A.J.T., Said Gattoufi, *Stochastic Cases of the Dynamic Job Shop Problem Based on the Genetic Algorithm to minimize Cmax*. Proceedings of 2017 4th International Conference onControl, Decision and Information Technologies (CoDIT’17), 2017: p. 0760-0765.

[11] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning* 1989: Addison-Wesley Longman Publishing Co., Inc.

[12] Abdelhafiez, A., E. Alba, and G. Luque, *Performance analysis of synchronous and asynchronous distributed genetic algorithms on multiprocessors*. Swarm and Evolutionary Computation, 2019. 49: p. 147-157.