Research Article

A Makespan Optimization Scheme for NP-Hard Gari Processing Job Scheduling Using Improved Genetic Algorithm

Adeyanju Sosimi,¹ Folorunso Oladimeji Ogunwolu,¹ and Taoreed Adegbola²

¹ Department of Systems Engineering, University of Lagos, Akoka, Yaba 23401, Nigeria

² Department of Mechanical Engineering, Tshwane University of Technology, Pretoria 0001, South Africa

Correspondence should be addressed to Adeyanju Sosimi; asosimi@unilag.edu.ng

Received 28 August 2013; Revised 17 December 2013; Accepted 18 December 2013; Published 6 April 2014

Academic Editor: Wen-Chiung Lee

Copyright © 2014 Adeyanju Sosimi et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

An optimization scheme for minimizing makespan of Gari processing jobs using improved initial population Genetic Algorithm (GA) is proposed. GA with initial population improved by using job sequencing and dispatching rules of First Come First Served (FCFS), Shortest Processing Time (SPT), Longest Processing Time (LPT), and Modified Johnson's Algorithm for m -machines in order to obtain better schedules than is affordable by GA with freely generated initial population and by individual traditional sequencing and dispatching rules was used. The traditional GA crossover and mutation operators as well as a custom-made remedial operator were used together with a hybrid of elitism and roulette wheel algorithms in the selection process based on job completion times. A test problem of 20 jobs with specified job processing and arrival times was simulated through the integral 5-process Gari production routine using the sequencing and dispatching rules, GA with freely generated initial population, and the improved GA. Comparisons based on performance measures such as optimal makespan, mean makespan, execution time, and solution improvement rate established the superiority of the improved initial population GA over the traditional sequencing and dispatching rules and freely generated initial population GA.

1. Introduction

One of the most important products obtained from the processing of cassava is "Gari." Gari is a creamy-white, granular flour with a slightly fermented flavor and a slightly sour taste made from fermented, gelatinized fresh cassava tubers. Gari processing industries occupy a substantial portion of small and medium enterprises (SMEs) in Nigeria. In the past few decades, research on Gari production has yielded tremendous gain particularly in the areas of developing Gari processing machine and improving on its quality. However, little or no attention has been given to scheduling of customers' orders in a way that would improve delivery performance and inventory management and reduce production cycle times and overall cost associated with the production process. Hence, operational bottlenecks are often experienced in the day to day activities while the desire for appropriate processor becomes imperative.

Job scheduling in a Gari processing firm is analogous to a flow shop in which a set of n -jobs have to be processed with identical flow patterns on m -machines. In their works [1–3] developed scheduling models on the following assumptions.

- (i) Each of the n -jobs has the same ordering of machines for its process sequence.
- (ii) At a time, every job is processed on one and only one machine, which means no job splitting.
- (iii) Each of the m -machines can process only one job at a time.
- (iv) The operations are not preemptible.
- (v) The operation processing times on the machines are known and fixed.
- (vi) Setup times of operations are independent of the sequences and therefore can be included in the processing time.

The Gari processing scheduling is an NP-hard problem [3] making it computationally difficult because with increasing number of jobs the computation time increases exponentially. There are no known algorithms for finding optimal solutions in polynomial time. Consequently, most researches are devoted to either simplifying the scheduling problem to the point where some algorithms can find solutions or devise efficient heuristics for finding better solutions. Motivated by the computational complexity of the problem, a makespan optimization scheme using genetic algorithm is presented here to develop a structure for the suitable scheduling of orders (jobs) in a Gari processing firm. In practice, the job shop scheduling has been approached mainly by using dispatching rules with the objective of finding a sequence that minimizes the makespan. For problems with 2 machines or 3 machines under specific constraints on job processing times, the efficient Johnson's algorithm [4] obtains an optimal solution for the problem. However, since this scheduling problem is NP-hard [5] the search for an optimal solution centres on more theoretical than practical importance. Therefore, since the 1960s a number of heuristics that provide near optimal or good solutions with limited computation effort have been proposed for flow shop sequencing. The performance of some earlier heuristics was evaluated in [6]. Heuristics can be classified according to two major categories: constructive or improvement methods. The constructive algorithms obtain directly a solution for the scheduling problem, that is, an n -job sequence, by using some procedure which assigns to each job a priority or an index in order to construct the solution sequence (see, e.g., [6–10]). An improvement method starts from a given initial solution and looks for a better one normally by using some neighborhood search procedure. Metaheuristics can also be considered as improvement heuristics. Among these groups of techniques are genetic algorithms (GAs), simulated annealing (SA), tabu search (TS), and their hybrids. The first proposed metaheuristics for the permutation flow shop scheduling problem (PFSP) are the simulated annealing algorithms by [11] demonstrating different tabu search approaches. Other algorithms are the path-based method of Werner [12] or the iterated local search (ILS) of Stutzle [13]. Recently, Rajendran and Ziegler [14] have proposed two very effective ant colony optimization (ACO) algorithms and Grabowski and Wodecki [15] a very fast TS approach. Ruiz and Maroto [16] give an updated and comprehensive review of flow shop heuristics and metaheuristics. Another recent review is given by Soewanda et al. [17] compared ant colony and combination of genetic algorithm and tabu search for solving flow shop scheduling problem. Nagano et al. [18] proposed a hybrid genetic algorithm, which combines genetic algorithm and NEH algorithm to solve similar problems. Ruiz et al. [19] proposed a robust genetic algorithm that considered some new genetic operators, population initialization, and generation on new population. In association with [20], Ruiz also proposed a robust hybrid-genetic algorithm. The paper discusses the comparison between ant colony, hybrid genetic, and robust-hybrid genetic algorithm. This paper focuses on the use of genetic algorithms (GA). A genetic algorithm is a computerized iterative search optimization technique. It

is based on the mechanics of natural selection and natural genetics that deals with populations of solutions rather than with a single solution. This type of algorithm provides near optimal schedules. The optimum value depends on the operators like crossover, mutation, number of iterations (generations), and so forth. In every generation, a new set of artificial individuals (strings) is created. Evolution of chromosomes over generations is by survival of the fittest. GA searches a problem space with a population of chromosomes and selects chromosomes for a continued search based on their performance. Each chromosome is decoded to form a solution in the problem space in the context of optimization problems. Genetic operators are applied to high performance structures (parents) in order to generate potentially fitter new structures (offsprings). Therefore, good performers propagate through the population from one generation to the next [20]. Holland [21] presented a basic GA called "simple genetic algorithm" in his studies that is described as in Algorithm 1.

A GA contains the following major ingredients: parameter setting, representation of a chromosome, initial population and population size, selection of parents, genetic operations, and a termination criterion.

This paper describes a makespan optimization scheme using a genetic algorithm with improved initial population as a structure for the suitable scheduling of orders (jobs) in a Gari processing firm. The makespan optimization scheme using GA has a number of new features compared to a traditional genetic algorithm. These include the generation and evolution of a 2-dimensional array of improved initial population size by incorporation of heuristics into initialization to generate well-adapted initial population. In this way, the genetic algorithm (GA) with elitism can guarantee to do no worse than the conventional heuristic does. In this paper, the initial populations are generated using traditional dispatching rules, with corresponding string numbers and string lengths analogous to the number of workstations and work orders. A hybrid of elitism and roulette wheel algorithms was used in the selection process based on job completion times decoded from chromosomes. The reproductive operators are the traditional crossover and mutation operators and a custom-made remedial operator. The performance of the formulated scheme using GA for the same test problem is then compared with some other sequencing and dispatching rules as well as GA scheme with freely generated initial population. A 20-job test problem was simulated through the integral 5-process Gari production routine.

The rest of this paper is organized as follows: Section 2 discusses the Gari processing system and scheduling framework, Section 3 presents detailed descriptions of the proposed method and algorithm, and Section 4 presents the data analysis. In Section 4, extensive comparison of the proposed method is undertaken with some other sequencing and dispatching rules.

Finally, in Section 5 some conclusions from the study are provided along with some future research directions.

ALGORITHM 1: Simple genetic algorithm ( ).

FIGURE 1: Gari processing flowchart.

2. The Gari Processing System

The Gari processing plant flowchart consists of the basic processes of peeling, grating, pressing, sieving, and roasting/drying (Figure 1). The firm that processes the cassava has one of each of these stage-wide machines. Thus, customers from all over the city arrive at the Gari processing firm at different periods of the day, with varying amounts of fresh cassava roots for processing.

In order to schedule the processing of customers' orders such that maximum profit is obtained, the principles guiding flow shop scheduling are adopted in which the cassava processing plant is considered as a 5-machine flow shop system where customers are free to bring their jobs at any time. However, each customer's order (fresh roots) passes through the machines in the same order. Since different quantities are brought for processing and the fresh roots have the same surface area characteristics, each order requires different amounts of processing time. In the current work, the unit of measurement is hour. Since test methods that could handle large numbers of orders are proposed here, the test problem considers a case of 20 customers. This corresponds to 20 individual jobs. The concern is in what way should the customers' orders be processed such that the firms profit is maximized? In order to solve this problem, the study considers a common measure of performance utilized in the flow shop scheduling literature—the makespan. The principle here is to monitor the completion time of the last scheduled customer's order. Much work has been done in understanding

this principle, as stated by [22]. There is much support for the use of the makespan approach as a measure of performance as evidenced by its extensive use in maximizing production rates and minimizing the mean idle time of machines [23].

3. Methodology

In this study, a GA scheme initialized by using job sequencing heuristics and dispatching rules of FCFS, SPT, LPT, and modified Johnson's algorithm to generate well adapted initial population for *Gari processing job scheduling* with makespan as the criterion is used. Before discussing how this improved GA is realized, the sequencing and dispatching rules used to improve the GA's initial population are discussed.

3.1. First Come First Served (FCFS) Rule. Using this rule, jobs are scheduled as they arrive for production. This is the traditional rule for scheduling.

3.2. Shortest Processing Time (SPT) Rule. Jobs are scheduled with this rule by sequencing them in ascending order of job processing times per process. The SPT scheme is depicted by Algorithm 2.

3.3. Longest Processing Time (LPT) Rule. Jobs are scheduled with this rule by sequencing them in descending order of job processing times per process.

```

Step 0. Read  $\{P_{ij}\}$  = the set of processing job times for the  $j$ th process
and  $\{J_i\}$  = the corresponding set of job numbers ( $1 \leq j \leq m$ ), ( $1 \leq i \leq n$ )
Step 1. Set  $i = 2$ ,  $j = k = 0$ 
Step 2. Set  $j = j + 1$ 
Step 3. Set  $k = k + 1$ 
Step 4. If  $P_{kj} > P_{ij}$  GO TO Step 5
Else Set  $V = P_{kj}$  and  $R = J_k$ , the corresponding job
      Set  $P_{kj} = P_{ij}$  and  $J_k = J_i$ 
      Then set  $P_{ij} = V$  and  $J_i = R$ 
Step 5. If  $k < n$ , GO TO Step 3
Else GO TO Step 6
Step 6. Set  $k = 0$ 
      If  $j < m$ , GO TO Step 2
      ELSE STOP

```

ALGORITHM 2: Shortest processing time (SPT) algorithm for n jobs, m processes.

To realize the equivalent longest processing time (LPT) algorithm, the premise of the conditional statement in Step 4 (If $P_{kj} > P_{ij}$) is changed to (If $P_{kj} < P_{ij}$) in the SPT Algorithm 2.

3.4. Modified Johnson's m -Machine Algorithm. Johnson's method is defined optimally for two machines. However, since it is optimal, and easy to compute, it is adaptable for 2 machines ($m = 2$). The idea is as follows: imagine that each job requires m operations in sequence, on $M_1, M_2, \dots, M_m$. We combine the first $m/2$ machines into an (imaginary) machining center, MC1, and the remaining machines into a machining center, MC2. Then the total processing time for a part on MC1 is equal to the sum of operation times on the first $m/2$ machines, and processing time for part on MC2 is equal to operation times on the last $m/2$ machines. By doing so, the m -Machine problem is reduced to a two-machining-center scheduling problem. The modified Johnson's algorithm is depicted in the immediate Algorithm 3.

3.5. Parameters Setting. The parameters in GA comprise the population size, number of generations, crossover probability, and mutation probability. These parameters are used as fixed values throughout the genetic evolution until the termination criterion is attained.

3.6. Encoding. In GA, each solution is usually encoded as a bit string. During the past years, many encoding methods have been proposed for scheduling problem [24]. Among various kinds of encoding methods, job-based encoding, machine-based encoding, and operation-based encoding methods are most often used for scheduling problems. This study adopts job-/machine-based encoding method. Each gene uniquely indicates an operation and can be determined according to its order of occurrence in the sequence. Let O_{ij} denote the j th operation of job i . The chromosomes can be translated into a unique list of ordered operations typified by the concatenated strings making up individual chromosomes as shown in Table 1.

3.7. The Algorithm

3.7.1. Generation of Initial Population. The initial population sets are generated using four traditional job dispatching rules: first come first served (FCFS), shortest processing time (SPT), longest processing time (LPT), and modified Johnson's algorithm.

The following are the GA parameters used:

- population size = 250
- number of strings = 5 (corresponding to the five stages of Gari processing)
- individual string length = 20 bytes (corresponding to the number of jobs used for the test problem here)
- crossover probability = 0.6
- mutation probability = 0.05
- number of generations = 500.

In GAs, these parameters are subjectively specified to fit the characteristics of the problem on hand and standard tested parameter ranges that research has established.

Step 1. Create a 2-dimensional array [250, 100] using random variations of the solution obtained with any of the traditional job sequencing and dispatching rules mentioned above (see Table 2).

Step 2. Using the corresponding job times for the job numbers in each byte position of each of the 5 strings in each chromosome, the makespan corresponding to each chromosome is computed according to the sequence in the chromosome.

Step 3. Arrange the reciprocal of the makespan (representing the fitness) obtained for the individual chromosomes of the initial population in descending order of magnitude.

Step 4. Obtain corresponding cumulative fitness value for each chromosome as ordered in Step 3.

TABLE 1: Job/Machine Based Encoding.

Chromosome 1	String 1, for Process 1	String 2, for Process 2	String 4, for Process 4	String 5, for Process 5
	$O_{1\ 1}, O_{9\ 1}, \dots, O_{3\ 1}$	$O_{7\ 2}, O_{6\ 2}, \dots, O_{4\ 2}$	$O_{14\ 4}, O_{5\ 4}, \dots, O_{3\ 4}$	$O_{1\ 5}, O_{8\ 5}, \dots, O_{4\ 5}$
Chromosome 2	String 1, for Process 1	String 2, for Process 2	String 4, for Process 4	String 5, for Process 5
	$O_{3\ 1}, O_{12\ 1}, \dots, O_{17\ 1}$	$O_{6\ 2}, O_{13\ 2}, \dots, O_{9\ 2}$	$O_{2\ 4}, O_{15\ 4}, \dots, O_{7\ 4}$	$O_{4\ 5}, O_{10\ 5}, \dots, O_{5\ 5}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Chromosome 250	String 1, for Process 1	String 2, for Process 2	String 4, for Process 4	String 5, for Process 5
	$O_{10\ 1}, O_{7\ 1}, \dots, O_{19\ 1}$	$O_{8\ 2}, O_{16\ 2}, \dots, O_{3\ 2}$	$O_{9\ 4}, O_{3\ 4}, \dots, O_{1\ 4}$	$O_{9\ 5}, O_{18\ 5}, \dots, O_{3\ 5}$

TABLE 2: Typical Array of Job Orders in Chromosomes.

	1–20	21–40	41–60	61–80	81–100
1	20, 1, 4, ..., 5	1, 5, 7, ..., 16	...	...	9, 3, 6, ..., 15
2	2, 13, 4, ..., 6	7, 9, 1, ..., 10	...	...	8, 4, 14, ..., 20
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
250	4, 1, 3, ..., 13	2, 1, 3, ..., 20	...	...	2, 11, 20, ..., 5

Step 5. Normalize the values of the cumulative fitness obtained and create class boundaries with them for roulette wheel selection.

3.7.2. Selection. At this stage selection of chromosomes that qualify as parents is made using a combination of roulette wheel selection and elitism.

Step 1 (selection by elitism). The chromosomes are classified into five distinctive groups of 50 from each of which the first and the last chromosomes in the ordered class are selected automatically to the next generation. Under Scheme 10 chromosomes are selected into the next generation.

Step 2 (selection using roulette wheel). Randomly generate pairs of integer numbers between 1 and 250 to determine parents for the genetic operation of reproduction. This scheme selects the balance of 240 chromosomes into the next generation.

3.7.3. Crossover. Crossover is an operation to generate new strings (i.e., children strings) from two parent strings. It is the main reproduction operator of GA. During the past years, various crossover operators had been proposed [25]. The work showed that the two-point crossover is effective for flow shop problems. Hence the two-point crossover method is used in this study.

The following steps are undertaken to execute the crossover operation.

Step 1. Generate a random number between 0 and 1 and compare it to the specified crossover probability. If the generated number is less than or equal to the specified probability, crossover is permitted; otherwise the pair goes by into the next generation.

Step 2. To determine the string where crossover will occur, generate random integers between 1 and 5; the numbers generated represent the strings where crossover will take place.

Step 3. To determine crossover positions, generate a random integer between 2 and 19 twice. And exchange the content of chromosome on each corresponding position.

3.7.4. Mutation. Mutation is another operator of GA. Such an operation can be viewed as a transition from a current solution to its neighborhood solution in a local search algorithm. It is used to prevent premature convergence and fall into local optimum. To determine mutation parents, generate two numbers independently between 1 and 250.

Step 1. Generate a random number between 0 and 1 to ascertain whether a parent will be mutated. If the number is less than the specified mutation probability, mutation is permitted; otherwise the pair of chromosomes selected goes by into the next generation.

Step 2. To determine the string where mutation will occur, generate a random integer between 1 and 5; the number generated represents the string where mutation will take place.

Step 3. To determine mutation position, generate a random integer between 1 and 20 and exchange the contents of the two chromosomes at the mutation position.

3.7.5. Remedial Operator. Remedial operation is necessitated after using the traditional GA operators as some numbers representing the jobs in strings are repeated. Hence, some between the ranges of 1 and 20 in which each number should occur once will be omitted. Thus, there is a need to rectify the situation. The scheme for remedial operator is explained below.

(1) Repetitions

Step 1. Search each string to determine if a value is repeated.

Step 2. Store the number of times it is repeated by indexing.

1	2		3			
3	7	3	"	"	3	16

Stage 1: (Conversion of m machines (or processes to 2-Machine/process equivalent)
 Step 0. **Set** $i = j = k = 0$
 Step 1. **Set** $k = k + 1$
 Step 2. **Set** $j = j + 1$
 Step 3. **Set** $i = i + 1$
 Step 4. **Read** P_{ij} (the processing time for the i th job on the j th machine/process) and corresponding job number J_i ($1 \leq j \leq m$) for each job i ($1 \leq i \leq n$)
 Step 5. $Q_{ik} = Q_{ik} + P_{ij}$ (update of the cumulative job time for the i th job on machine $k = 1, 2$)
 Step 6. **If** $i < n$ GO TO Step 3
Else Set $i = 0$ and GO TO Step 7
 Step 7. **If** $j < m/2$ GO TO Step 3
     If $j = m/2$ GO TO Step 1
     If $m/2 < j < m$ GO TO Step 2
     Elseif $j = m$, STOP

 Stage 2: 2-machine (process) equivalent Scheduling Algorithm
 Step 8. **Read** $S = \{Q_{ik}\}$ for all $i, k; 1 \leq i \leq n, k = 1, 2$
 Step 9. **Set** $r = 0, s = n + 1$
 Step 10. **Read** $P_{ik} = \min_{i, k} Q_{ik}$ for all $i, k \in S$
     If $k = 1$, set $r = r + 1$
     Set job J_i with time P_{ik} as Scheduled job J_r
Else Set $s = s - 1$
     Set job J_i with time P_{ik} as Scheduled job J_s
 Step 11. Set $S = S - P_{i1} - P_{i2}$
 Step 12. **If** $S = \emptyset$ (empty) STOP
     Else GO TO Step 10

ALGORITHM 3: Johnson's m -machine scheduling algorithm.

Step 3. Generate a random integer between 1 and the number of times the number is repeated.

Step 4. The value generated represents the index of the repeated value that will be retained.

(2) Omission

Step 1. Search each string to determine if a point value is omitted.

Step 2. Store the numbers omitted by indexing.

Step 3. Generate the numbers that are omitted randomly. The value represents the omitted value that a vacant point will be assigned.

Step 4. Generate the index of the vacant position randomly. The value generated represents the point where the value generated in Step 3 will be assigned.

3.7.6. *Evaluation.* The makespan will be evaluated, and the corresponding value will be stored. The process will be repeated. If a better optimal solution is obtained, it will replace the penultimate optimal value.

3.7.7. *Termination.* GA continues to process the above procedure until attaining the stop criterion set by user. The commonly used criteria are as follows: (1) the number of executed generations; (2) a particular object; and (3) the homogeneity of population. This study uses 500 generations to serve as the termination condition.

3.8. *Test Problem.* A 20-job test problem was simulated through the integral 5-process Gari production routine. As exemplified in our previous discussion the goal is to determine appropriate job order that will guarantee productivity and time response to customers order without delay. In the real world jobs are described by arrival time, processing time, and due-dates. In this work only arrival time and processing times of jobs (for peeling processing time—PPT, grating processing time—GPT, pressing processing time—PRPT, sieving processing time—SPT, and frying processing time—FrPT) were considered. Table 3 shows the job data.

3.9. *Method of Solution.* All algorithms were coded and run within the MATLAB environment. The test problem was run with a MATLAB 7.5 program using the algorithm to evaluate the makespan for each of the chromosomes. The hardware used for running the M-file programme is a 1.67 GHz Intel Core Duo CPU T2300 with 512 MB memory.

TABLE 3: Data on Gari processing job flow.

Job number (i)	Job arrival/h	Volume (tons) V_i	Job due time/hr	PPT	GPT	PRPT	SPT	FrPT
1	13	8	21	5.92	1.44	2.24	1.28	1.76
2	10	10	25	7.4	1.8	2.8	1.6	2.2
3	33	8	56	5.92	1.44	2.24	1.28	1.76
4	38	7	48	5.18	1.26	1.96	1.12	1.54
5	16	2	37	1.48	0.36	0.56	0.32	0.44
6	88	5	100	3.7	0.9	1.4	0.8	1.1
7	16	7	112	5.18	1.26	1.96	1.12	1.54
8	31	12	50	8.88	2.16	3.36	1.92	2.64
9	58	5	67	3.7	0.9	1.4	0.8	1.1
10	79	5	142	3.7	0.9	1.4	0.8	1.1
11	64	10	88	7.4	1.8	2.8	1.6	2.2
12	49	6	116	4.44	1.08	1.68	0.96	1.32
13	57	5	95	3.7	0.9	1.4	0.8	1.1
14	18	10	58	7.4	1.8	2.8	1.6	2.2
15	128	10	140	7.4	1.8	2.8	1.6	2.2
16	88	9	116	6.66	1.62	2.52	1.44	1.98
17	90	5	142	3.7	0.9	1.4	0.8	1.1
18	95	7	115	5.18	1.26	1.96	1.12	1.54
19	129	8	156	5.92	1.44	2.24	1.28	1.76
20	23	4	46	2.96	0.72	1.12	0.64	0.88

3.10. Methods of Analysis. The results obtained by using different sequencing and dispatching rules to generate initial population for the genetic algorithm optimization scheme are compared with the results from the GA scheme using freely generated initial population and those using the traditional sequencing and dispatching rules of shortest processing time, longest processing time, first come first served, and the modified Johnson's algorithm. The bases of comparison of results are the optimal mean-makespans obtained from programme execution times for the various options.

The Mean-Makespan. The makespan for a set of jobs under consideration is the total time durations through all the five integral production processes of all jobs received within a time horizon in the Gari processing job shop. The mean-makespan is thus the quotient of the makespan and the number of jobs processed. The optimal mean-makespan is functional defined as

$$T = \frac{\sum_{i=1}^n t_i^{\min}}{n}, \quad (1)$$

where $t_i^{\min}$ is the minimum time (span or duration) for job i to be completed through the five production processes.

The $t_i^{\min}$ is also equivalent to the minimum difference between the time point of completion of job i and the arrival time point in the job shop. Assuming there is no breakdown within the system of production, this span is dependent on the sequence of jobs at each stage or process of production. For the traditional sequence and dispatching rules the sequences are well defined. For the new approach used in this work, an improved solution with sequences

TABLE 4: Optimal makespan running the GA without improved initial population.

Population size	Execution time	Optimal makespan
250	17.9	252.86
500	24.141	243.16
1000	32.53	235.78
5000	121.188	236.18
10000	337.297	230.42
20000	1068.2	228.62
50000	5550	226.94
100000	6720	215.42

evolving through genetic algorithm optimality process is sought.

Execution Time. As a basis for practical implementation, the time of execution of a sequence of processes of the sort used in this work is paramount. If a particular scheme is to be more beneficial than any other to industrial practitioners who are time-conscious, it must be executable faster. This informs the comparison of the execution time for the different options of scheduling used in this work.

Population Variation within the Genetic Algorithm (GA) Scheme. Because of the subjective nature of choice of GA parameters, this work attempted to vary the population sizes of the genetic algorithm scheme used and compare both the mean-makespans and execution time for varying GA populations.

TABLE 5: Optimal makespan with traditional job sequencing and dispatching rules.

Sequencing rule	Execution time	Optimal makespan
FIFO	15.67	148.05
SPT	16.99	149.04
LPT	16.76	238.06
Johnson's algorithm	18.22	269.06

TABLE 6: Optimal makespan for running the GA with improved initial population.

Population size	Execution time	Optimal makespan
250.00	19.50	154.27
500.00	26.80	153.70
1000.00	34.73	152.02
5000.00	151.47	148.03
10000.00	270.59	148.01
20000.00	963.87	146.80
50000.00	3851.22	146.55
100000.00	6211.59	145.31

4. Results and Analysis

Table 4 clearly depicts the result obtained when the test problem was addressed using freely generated initial GA population. Table 5 indicates the result obtained using the four traditional job sequencing and dispatching methods as elucidated in Section 3. A mixed breed of the results obtained as in Table 5 indicates that the various job orders constitute the family initial population. The results obtained for running the algorithm with improved 2-dimensional array initial GA population that were generated from the traditional job sequencing and dispatching rules such as FCFS, SPT, LPT, and modified Johnson's algorithm are shown in Table 6.

From Tables 4–6 it can be deduced that when the algorithm was tested with an improved initial GA population a better optimal makespan was realized as against the freely generated initial GA population. Generally, it can be stated that when the algorithm was tested with improved initial GA population a large difference was observed in the execution time and optimal makespan. Graphical illustrations of the above analysis are shown in Figures 2–4 utilizing the data in Tables 4–6.

Figure 2 shows that when heuristics is incorporated in the generation of initial population using traditional dispatching rule, a better optimal makespan is obtained. The result also indicates that as the size of the initial population increases the corresponding execution time also increases but the execution time obtained with a freely generated initial GA population is quite high when compared with the execution time obtained using the improved initial GA population.

From Figure 3, by comparing optimal makespans obtained, better optimal makespans are obtained with improved initial GA population than with randomly (freely) generated initial population. A better optimal makespan is obtained with improved initial GA population. Both plots

FIGURE 2: Execution time against population size.

FIGURE 3: Optimal makespan against population size (with improved initial GA population and freely generated initial population).

are downward sloping from left to right and the population size increases while a better optimal makespan is obtained. The improved initial population proves to be much more effective.

It can be deduced from Figures 4 and 5 that a better optimal makespan is obtained with the improved initial GA population.

Table 7 shows the improvement rate in optimal makespan when the algorithm was run with improved initial population over freely generated initial population; a remarkable improvement was observed when the population size was of 250 chromosomes. The average improvement rate of improved initial population over freely generated initial population is 0.360. This further confirms the superiority of the method over the other methods considered in this work.

TABLE 7: Improvement Rate of Improved GA over Freely generated GA.

Population size (chromosome)	Optimal makespan freely generated initial population (hrs.)	Optimal makespan improved initial population (hrs.)	Improvement rate (%)
250	252.86	154.27	0.390
500	243.16	153.70	0.368
1000	235.78	152.02	0.355
5000	236.18	148.03	0.373
10000	230.42	148.01	0.358
20000	228.62	146.80	0.358
50000	226.94	146.55	0.354
100000	215.42	145.31	0.325

FIGURE 4: Mean makespan against population (with freely generated initial GA population).

FIGURE 5: Mean makespan against population (improved initial GA population).

5. Conclusion

This study developed a makespan optimization scheme using genetic algorithm with improved initial population as a structure for the suitable scheduling of orders (jobs) in a Gari processing firm using initial GA population based on job sequencing and dispatching rules. A number of new features were incorporated into the improved initial GA population. Several yardsticks were used for quantifying the performance of the scheme; the computed results are then compared when the same test problem is addressed using a freely generated GA initial population and four traditional job dispatching rules as follows: first come first served (FCFS), shortest processing time (SPT), longest processing time (LPT), and modified Johnson's algorithm. Analyses based on the computed results show that the method can propitiously improve the results obtained by using freely generated initial GA population and traditional job sequencing and dispatching rules. A better optimal makespan is obtained with the improved initial GA population and as the size of the initial population increases, the corresponding execution time increases; the execution time obtained with a freely generated initial GA population is quite high when compared with the execution

time obtained using the improved initial GA population. In conclusion, GA provides a variety of options and parameter settings which still have to be fully investigated. This study has demonstrated that scheduling of orders (jobs) in a Gari processing firm can be addressed by means of GA and suggests that such procedures are well worth exploring in the context of solving large and difficult combinatorial problem which may be of interest to industrial practitioners and academic researchers in the field of evolutionary computing and machine scheduling.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

- [1] T. Aldowaisan and A. Allahverdi, "New heuristics for no-wait flowshops to minimize makespan," *Computers and Operations Research*, vol. 30, no. 8, pp. 1219–1231, 2003.

- [2] R. Bellman, "Mathematical aspects of scheduling theory," *Journal of the Society for Industrial and Applied Mathematics*, vol. 4, no. 3, pp. 168–205, 1956.
- [3] M. Y. Wang, S. P. Sethi, and S. L. Van De Velde, "Minimizing makespan in a class of re-entrant shops," *Operations Research*, vol. 45, no. 5, pp. 702–712, 1997.
- [4] C. H. Campbell, D. R. Dudek, and S. M. Smith, "A heuristic algorithm for n job, m machine sequencing problem," *Management Science*, vol. 16, no. 10, pp. 630–637, 1970.
- [5] P.-C. Chang, S.-H. Chen, and K.-L. Lin, "Two-phase sub population genetic algorithm for parallel machine-scheduling problem," *Expert Systems with Applications*, vol. 29, no. 3, pp. 705–712, 2005.
- [6] D. G. Dannenbring, "An evaluation of flow-shop sequencing heuristics," *Management Science*, vol. 23, no. 11, pp. 1174–1182, 1977.
- [7] C.-L. Chen, V. S. Vempati, and N. Aljaber, "An application of genetic algorithms for flow shop problems," *European Journal of Operational Research*, vol. 80, no. 2, pp. 389–396, 1995.
- [8] H. D. Pour, "A new heuristic for the n-job, m-machine flow-shop problem," *Production Planning and Control*, vol. 12, no. 7, pp. 648–653, 2001.
- [9] A. Dolgui, B. Finel, N. Guschinsky, G. Levin, and F. Vernadat, "MIP approach to balancing transfer lines with blocks of parallel operations," *IIE Transactions*, vol. 38, no. 10, pp. 869–882, 2006.
- [10] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, Mass, USA, 1989.
- [11] S. Reza Hejazi and S. Saghaian, "Flowshop-scheduling problems with makespan criterion: a review," *International Journal of Production Research*, vol. 43, no. 14, pp. 2895–2929, 2005.
- [12] F. Werner, "On the heuristic solution of the permutation flow-shop problem by path algorithms," *Computer and Operations Research*, vol. 20, no. 7, pp. 707–722, 1993.
- [13] T. Stutzle, *Local search algorithm for combinatorial problems: analysis, improvements, and new applications [Ph.D. thesis]*, Darmstadt University of Technology, Department of Computer Science, 1998.
- [14] C. Rajendra and H. Ziegler, "Ant colony algorithms for permutation flowshop scheduling to minimize makespan/total flow-time of jobs," *European Journal of Operational Research*, vol. 155, no. 2, pp. 426–438.
- [15] J. Grabowski and M. Wodecki, "A very fast tabu search algorithm for permutation flow shop problem with makespan criterion," *Computers and Operations Research*, vol. 31, no. 11, pp. 1891–1909, 2004.
- [16] R. Ruiz and C. Maroto, "A comprehensive review and evaluation of permutation flowshop heuristics," *European Journal of Operational Research*, vol. 165, no. 2, pp. 479–494, 2005.
- [17] J. Soewande, T. Octavia, and I. H. Sahputra, "Robust hybrid genetic algorithm for a flow-shop scheduling problem (A case study at PT FSCM manufacturing Indonesia)," *Jurnal of Teknik Industri*, vol. 9, no. 2, pp. 144–151, 2008.
- [18] M. S. Nagano, R. Ruiz, and L. A. N. Loren, "A constructive genetic algorithm for permutation flow shop scheduling," *Computer and Industrial Engineering*, vol. 55, no. 1, pp. 195–207, 2008.
- [19] R. Ruiz, J. C. Garcia-Diaz, and C. Maroto, "Considering scheduling and preventive maintenance in the flow shop sequencing problem," *Computer and Operations Research*, vol. 34, no. 11, pp. 3314–3330, 2007.
- [20] A. El-Bouri, N. Azizi, and S. A. Zolfaghari, "Comparative study of a new heuristic based on adaptive memory programming and simulated annealing: the case of job shop scheduling," *European Journal of Operational Research*, vol. 177, no. 3, pp. 1894–1910, 2007.
- [21] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*, University of Michigan Press, 1975.
- [22] J. C.-H. Pan and J.-S. Chen, "Minimizing makespan in re-entrant permutation flow-shops," *Journal of the Operational Research Society*, vol. 54, no. 6, pp. 642–653, 2003.
- [23] J. C. H. Pan and J. S. Chen, "A comparative study of schedule-generation procedures for the re-entrant shops," *International Journal of Industrial Engineering*, vol. 11, no. 4, pp. 313–321, 2004.
- [24] M. Pinedo, *Scheduling Theory, Algorithms and Systems*, Prentice Hall, Englewood Cliffs, NJ, USA, 2nd edition, 2002.
- [25] S. G. Ponnambalam, P. Aravindan, and S. Chandrasekaran, "Constructive and improvement flow shop scheduling heuristics: an extensive evaluation," *Production Planning and Control*, vol. 12, no. 4, pp. 335–344, 2001.