2023 IEEE Symposium Series on Computational Intelligence (SSCI)
Mexico City, Mexico. December 5-8, 2023

Quantum Representation Based Job Shop Scheduling

Kazi Shah Nawaz Ripon
Oslo Metropolitan University
Oslo, Norway
karip8799@oslomet.no

Ashay Singh
Ostfold University College
Halden, Norway
ashay.singh@hiof.no

***Abstract*—This paper proposes a quantum representation-based genetic algorithm for solving the job-shop scheduling problem, aiming to minimize the makespan. The job-shop scheduling is a typical scheduling problem that falls under the NP-hard combinatorial optimization problems and has undergone extensive investigation in the literature. Over time, various heuristic and intelligent methods have been developed to tackle this challenging problem. Inspired by the promise of quantum computing, this paper explores using quantum information representation and processing techniques to enhance the performance of conventional genetic algorithms on classical computers to solve the job-shop scheduling problem. The proposed quantum-inspired genetic algorithm employs a conversion mechanism of quantum representation to code the schedule; and utilizes a rotation angle table to update the population. The effectiveness of the quantum-inspired genetic algorithm is compared to that of a standard genetic algorithm, with experimental results confirming the potential of the proposed approach in tackling complex combinatorial optimization problems.**

***Index Terms*—quantum genetic algorithm, job-shop scheduling problem, converted quantum representation, rotation angle table**

I. Introduction

Scheduling aims to efficiently allocate shared resources, such as machines or personnel, over a specified period to complete multiple tasks or jobs while adhering to predetermined constraints. Typically, the process of scheduling involves solving an optimization or search problem by simultaneously arranging time, space, and often scarce resources. One of the most challenging scheduling problems is the job shop scheduling problem (JSSP), which consists of scheduling n distinct jobs on m identical machines. Each job must be processed on a set of m machines, one after the other, following a specific technological sequence. The jobs need to be scheduled without interruption, cancellation, or without prior knowledge of the upcoming jobs. The complexity of JSSP escalates with the number of constraints and the search space's size, making it one of the most difficult combinatorial optimization problems [1].

Due to industrial automation's growth, real-world JSSPs entail more jobs, machines, and additional constraints and flexibilities. Consequently, the JSSP has become a practical issue in the manufacturing industry. As a result, both the research community and the manufacturing sector have shown substantial interest in addressing this issue. As real-world JSSPs become larger and more complex, exact methods like dynamic programming or branch-and-bound become computationally expensive, especially when aiming to find optimal solutions [2]. Therefore, obtaining near-optimal solutions using stochastic techniques such as evolutionary algorithms (EAs) is more practical and feasible. The existing literature justifies the effectiveness and robustness of EAs in solving practical combinatorial problems [3]–[5].

Genetic algorithms (GAs) belong to the family of EAs, which draw inspiration from natural selection. They work with a *population* of solutions and use evolution operators, such as selection, mutation, or recombination, to improve the quality of the solutions. GAs provide effective and robust optimization and search techniques for large-scale combinatorial optimization problems. Since the first coming of the GA [6], the field of scheduling has seen a significant amount of research utilizing GAs, with various methods proposed in the literature to tackle the JSSP [1], [2], [7]–[9]. However, GAs still have limitations regarding low searching efficiency, slow convergence speed, and susceptibility to premature convergence [10]. Researchers have therefore focused on addressing these shortcomings by introducing various improvement methods. This paper proposes a quantum representation-based genetic algorithm combining GAs and quantum theory in this context.

Quantum algorithms utilize quantum mechanics principles to achieve efficient computation. This efficiency is evident when the algorithm is executed on a quantum computer, while performing the same operation on a classical computer may consume considerable resources [11]. Quantum-inspired computation utilizes the principles and concepts of quantum mechanics, like qubits, superposition, quantum gates, or quantum measurement, to solve various problems in a classical computing environment [12]. Recent advancements in quantum technology have demonstrated that for specific problems, especially for polynomial-time problems like factoring [13] or searching in an unstructured database [14], quantum computers can provide a significant advantage over classical computers. Given the exceedingly large search in most real-world problems tackled by GAs, investigating how GAs can leverage quantum parallelism on classical computers can be an intriguing research avenue.

Since the late 1990s, researchers have been exploring the potential interaction between quantum computing and evolutionary computation [15]. While quantum-inspired GA is based on quantum computing and quantum mechanics, it is not a quantum algorithm and was proposed for classical computers rather than quantum mechanical hardware [16], [17]. The quantum-inspired GA provides advantages in developing quantum phenomena such as entanglement or superposition. Here, the smallest unit of information is a *qubit* rather than a classical bit. Unlike classical bits, the *qubit* represents a 0 and 1 superposition. An N–*qubit* chromosome is a string of N–*qubits* representing a linear superposition of binary states in the probabilistic sense. Unlike the classical population, where individuals can only represent one potential solution, each "individual" in a quantum population is a superposition of multiple possible solutions. Consequently, the *qubit* individual guarantees higher population diversity than other known representations [16]. Thus, the quantum-inspired GA offers an advantage in population diversity, as quantum populations can be exponentially larger than classical populations of the same size.

Despite the increasing importance and diversity of scientific literature on quantum solutions for hard combinatorial optimization problems, there are still very few studies on the JSSP [18]. The limited existing research on the JSSP mainly consists of "real" quantum algorithms designed for use on quantum processors [19], [20]. This paper, however, explores the feasibility of using a quantum representation-based genetic algorithm (QGA) to solve the JSSP on classical computers. Drawing on ideas from [21], this work employs a quantum representation of the population, achieved through a conversion mechanism and a similar rotation table for population updates. The objective of the QGA is to minimize the makespan of the JSSP, defined as the maximum completion time of all jobs. The experimental results on several benchmark JSSP datasets of varying sizes indicate that the proposed QGA is effective in reaching the optimal schedule compared to the standard genetic algorithm (sGA). It is imperative to note that the primary objective of this study is to investigate the viability of quantum effects within GAs for addressing combinatorial optimization problems, emphasizing feasibility rather than an exclusive focus on attained performance.

This paper is structured as follows. Section II provides background on the JSSP and quantum computing. Section III explains the proposed QGA for solving the JSSP, including quantum representation of chromosomes and the effect of rotation gates on chromosomes. Section IV presents the experimental setup, results, and discussion, while Section V concludes the paper and outlines potential future research.

II. Background

This section introduces the concepts associated with JSSP and provides an overview of quantum computing and quantum GA.

A. The Job Shop Scheduling Problem (JSSP)

In a traditional nxm JSSP, there are n separate jobs ($\{J_j\}_{1\leq j\leq n}$), that must be executed using m machines ($\{M_k\}_{1\leq k\leq m}$). The objective is to find the *optimum order* in which the jobs should be processed, aiming to minimize specific performance measures. These measures could include the overall time needed to finish all jobs (makespan), the average duration of jobs from start to finish (mean flow time), or the average delay of jobs from their scheduled deadlines (mean tardiness). Every job J_j requires an uninterrupted processing on each machine M_k. The processing of job J_j on machine M_k is identified as an operation O_{jk}. The duration of each operation is continuous and exclusive to a machine for a duration of t_{jk}, which is its processing time. A technological sequence of x_j operations is assigned to each job. Table I illustrates a 4x3 JSSP, where Job-1 undergoes processing on Machine-3 for a duration of 4 unit and then by Machine-1 for 3 units, and so forth.

TABLE I: A 4 x 3 JSSP

	(k,t)	(k,t)	(k,t)
Job-1	3,4	1,3	2,6
Job-2	2,8	3,5	1,4
Job-3	1,5	3,4	2,2
Job-4	2,4	1,4	3,2

The JSSP poses a significant difficulty as a combinatorial optimization problem. The task of minimizing makespan in even a simple version of the JSSP becomes NP-hard when involving more than two machines. The search space for the standard JSSP is $n!^m$, making it impractical to explore each potential solution due to the exponential rise in computational time needed with larger problem sizes.

B. Quantum Inspired Genetic Algorithm

In the context of quantum GA, *qubits* are utilized for encoding genetic individuals, Q-gates are applied to manipulate *qubits* and produce offspring, and a probabilistic observation process connects genotypes and phenotypes [11]. The state of a *qubit* can be characterized by (1).

$$\psi = \alpha\left|0\right\rangle + \beta\left|1\right\rangle \tag{1}$$

Where the values $|0\rangle$ and $|0\rangle$ correspond to classical bit values of 0 and 1, respectively. The complex numbers α and β represent the probability amplitudes associated with the "0" and "1" states, respectively. When a quantum particle is observed, it takes on one and only one state in the measurement basis (either $|0\rangle$ or $|1\rangle$) [15]. Similarly, When a *qubit* is in a state of superposition, it exists in both states simultaneously but collapses to one state upon observation [22]. Thus, upon observation, the values $|\alpha|^2$ and $|\beta|^2$ correspond to the probabilities that the *qubit* is in the '0' or '1' state. Hence, the condition is met for the normalized state (2).

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2}$$

In quantum GA, a chromosome is composed of N *qubits*, and its state vector contains all the information for the quantum system. For instance, if there are two *qubits*, they can be in any one of four possible states: $|00\rangle$, $|01\rangle$, $|10\rangle$, or $|11\rangle$. Because *qubits* exist in a superposition state until observed, a chromosome with p *qubits* can represent 2^p states simultaneously. In contrast, a classical GA would require chromosomes that are 2^p bits long to represent the same number of states.

The traditional quantum GAs deploy a quantum gate to replace the crossover operation used in GAs. The phase rotation gate (3) is the most basic type of quantum gate. When it is applied to the initial state vector ψ in (1), a new state vector ψ' is generated, as shown in (4). The scope of quantum GA can also be expanded by incorporating other evolutionary operators, like quantum mutation [23], along with implementing parallelization techniques [24], which can reduce the overall computational time.

$$R(\theta) = \begin{matrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{matrix} \tag{3}$$

$$R(\theta|\psi') = \begin{matrix} cos(\theta_0 + \theta) \\ sin(\theta_0 + \theta) \end{matrix} \tag{4}$$

III. Quantum Genetic Algorithm for JSSP

This section elaborates on the quantum chromosome representation, the impact of the rotation gate on the chromosome, the operational procedure of the schedule builder, and the genetic operators used in the proposed QGA.

A. Quantum Chromosome Representation

The primary advantage of quantum chromosomal representation is the ability to utilize quantum parallelism, which enables the recording and simultaneous searching of multiple solutions. This feature can accelerate achieving the global optimal solution and improve optimization compared to typical GA representations [25]. This work employs an indirect chromosome representation for solving the JSSP. The binary format is used to mimic the quantum representation of *qubits*. However, the schedule builder, mutation, and crossover operations necessitate decimal numbers. Hence, the initial quantum population is converted to decimal format.

Considering the example in Table I, which pertains to a JSSP with 4 (*n*) Jobs and 3 (*m*) Machines, a single chromosome representing one complete schedule should consist of 12 decimal digits. For instance:

$$\text{Chrom–1} = 1\ 2\ 4\ 2\ 2\ 3\ 1\ 3\ 4\ 1\ 3\ 4 \tag{5}$$

To convert a decimal value of any single gene into binary, we need $\lceil \log_2 n + 1 \rceil$ bits. Considering the chromosome in (5), it would be $\lceil \log_2 4 + 1 \rceil$ = 3 bits. Consequently, for a chromosome consisting of 12 genes (decimal values), we would require $12*3$ = 36 bits to represent it in binary format.

To expedite the generation of the 36 bits representing a complete chromosome, smaller gene sequences are generated individually for each machine and combined to obtain the final chromosome. Instead of generating 36 bits all at once, 12 bits are generated in parallel for each machine, and these segments are subsequently merged to form the complete chromosome. The following steps (1 to 8) outline the procedure for generating the gene sequences in quantum representation and converting them into decimal format. Finally, in step 9, these sequences are combined to produce a single decimal sequence, which can be considered one *Final Chromosome* (*C*) within the initial population, as shown in (6).

$$\begin{aligned} C = \text{gene sequence } M_1 + \text{gene sequence } M_2 \\ + \text{gene sequence } M_3 \end{aligned} \tag{6}$$

Step 1: An initial binary string of length $n * \lceil \log_2 n + 1 \rceil$ with all zeros is generated to represent all the gene sequence values for one machine. Considering the chromosome in (5), the total number of bits required to represent one complete job sequence for one machine (G) = the number of jobs $*$ bits needed for one job = $4 * 3$ = 12 bits. Hence,

G = 0 0 0 0 0 0 0 0 0 0 0 0

Step 2: A two-dimensional array stores the probability amplitudes (α and β) for all positions, as depicted in Fig. 1. This array holds the probability amplitudes of the 12 bits for quantum operations. This step aims to simulate the standard binary bits as *qubits* by assigning them probability amplitudes α and β.

α_1	α_2	α_3	α_4	α_5	α_6	$\cdots$	$\cdots$	$\cdots$	α_{12}
β_1	β_2	β_3	β_4	β_5	β_6	$\cdots$	$\cdots$	$\cdots$	β_{12}

Fig. 1: Quantum chromosome structure

Step 3: To achieve an equal superposition state for the *qubits*, all amplitude values are initialized to $\frac{1}{\sqrt{2}}$, as explained in the Hadamard Gate [26]. It ensures that the probability of all quantum superposition states is identical. This is done by applying a rotation angle (θ) to the probability amplitudes, resulting in modified amplitudes α', β', as shown in (7).

$$\begin{pmatrix} \alpha_i' \\ \beta_i' \end{pmatrix} = R(\theta) \begin{pmatrix} \alpha_i \\ \beta_i \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{pmatrix} \alpha_i \\ \beta_i \end{pmatrix} \tag{7}$$

Step 4: The rotation gate is applied to each column of a chromosome, which represents a bit value, using a rotation angle θ ($0.02\pi \leq \theta \leq 0.05\pi$) and a unit vector n along the axis of rotation. This results in a new set of probability amplitude values. This process ensures that the final values of the columns are not all the same, introducing some variation among them. Let's assume that Fig. 2 represents the modified values after applying the rotation gate. In this modified state, the sum of the squares of each column value adds up to 1, as stated in (2). These amplitude values are stored for future use in subsequent steps as well.

0.707107	0.707107	$\cdots$	$\cdots$	−0.707107	−0.866025
0.707107	−0.707107	$\cdots$	$\cdots$	0.707107	0.500000

Fig. 2: Modified probability amplitudes

Step 5: Next, a random number, r, is selected within the range of 0 to 1. The value of r is then compared with the i^{th} index of the modified probability amplitude in Fig. 2. If $|\alpha_i|^2$ is greater than r, the corresponding bit (G_i) in the gene subsequence G (which initially consisted of all zeros) is assigned a value of 1; otherwise, it remains as 0. For instance, in Fig. 2, this process would be applied as follows:

Let, r_1 = 0.4. Since, $|\alpha_1|^2 \geq r_1$, G$_1$ is set to 1. This step is repeated for all 12 bits to obtain the final binary string consisting of zeros and ones; and, finally, generates the gene sequence for Machine–1 (M_1). For example,

$$\text{G} = 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \longrightarrow 1\,0\,1\,1\,0\,0\,0\,0\,1\,1\,1\,0$$

Step 6: Afterwards, the gene sequence is divided into sections of 3 ($\lceil \log_2 n + 1 \rceil$) bits each, and subsequently converting these sections into decimal values.

$$1\,0\,1\,1\,0\,0\,0\,0\,1\,1\,1\,0 \longrightarrow 1\,0\,1 \,|\, 1\,0\,0 \,|\, 0\,0\,1 \,|\, 1\,1\,0 \longrightarrow 5\,4\,1\,6$$

Step 7: The decimal values of each section are subjected to the modulus with n (job) to ensure that no values exceed the number of jobs for that particular JSSP. Additionally, the resulting values are incremented by 1 to ensure that the smallest value remains 1 instead of 0.

$$5\ \%\ 4,\ 4\ \%\ 4,\ 1\ \%\ 4,\ 6\ \%\ 4 \longrightarrow 1+1,\ 0+1,\ 1+1,\ 1+1 \\ \longrightarrow 2\,1\,2\,3$$

Step 8: The previous step generates a decimal sequence ranging from 1 to n, allowing for possible repetitions. To eliminate duplicates in the sequence, unique values are chosen from the range (1, n), and any duplicates are substituted with missing values within that range. These replaced values are then appended at the end of the sequence to form a valid gene sequence. In the context of the JSSP, a valid chromosome is characterized by the occurrence of each job only once and the inclusion of all jobs.

Taking the above gene sequence for M_1 as an example, there are two occurrences of '2'. The second '2' is removed and replaced by the first missing job, which is '4' for this machine. Accordingly, Job-4 is added at the end, and a valid job sequence for M_1 is generated.

$$\mathbf{2}\ 1\ \mathbf{2}\ 3 \longrightarrow \mathbf{2}\ 1\ \mathbf{-}\ 3 \longrightarrow 2\ 1\ 3\ \mathbf{4}$$

Step 9: The steps from 1 through 8 are repeated for M_2 and M_3 to obtain their final valid gene sequences. Finally, following (6), all these sequences are combined to generate one initial valid chromosome. Assuming that the sequence for M_2 is (4 1 2 3), and M_3 is (3 2 1 4) after repeating the steps, the one final valid chromosome (C_1) for the 4–job, 3–machine JSSP is shown below. It is used as one of the initial chromosomes in the population.

$$C_1 = 2\,1\,3\,4 \quad 4\,1\,2\,3 \quad 3\,2\,1\,4$$

After the sequence is finalized, it is sent to the *schedule builder* (discussed in Section III-B) to assess its fitness and to estimate the makespan. In this process, the best makespan is recorded for later use in the rotation gate. The use of rotation gates aids in the development of the solution. After a generation concludes, the population undergoes adjustments based on the guidelines outlined in the look-up table (Table II), following the recommendations in [21] and implementing the quantum rotation gate as defined in (7).

TABLE II: Look-up table to decide the rotation angle for the individual [21]

p_i	b_i	$f(\text{p}) < f(\text{b})$	$\Delta\theta_i$	$s(\alpha_i, \beta_i)$	
				$\alpha_i\beta_i > 0$	$\alpha_i\beta_i < 0$
0	0	F	0.2π	-1	$+1$
0	0	T	0	0	0
0	1	F	0.5π	$+1$	-1
0	1	T	0	0	0
1	0	F	0.5π	-1	$+1$
1	0	T	0	0	0
1	1	F	0.2π	$+1$	-1
1	1	T	0	0	0

In this context, p_i refers to the i^{th} bit of the current individual, while b_i represents the i^{th} bit of the best individual obtained in one generation. The fitness of an individual, denoted as $f(a)$, is determined by its makespan value, with a lower value indicating greater fitness. The rotation angle for the i^{th} bit is represented by $\Delta\theta_i$. The sign of the i^{th} probability amplitudes, after their multiplication, is denoted by $s(\alpha_i, \beta_i)$. It is worth noting that the probability amplitudes stored in Step-4 play a crucial role in this process.

After the first generation, the best individual (b) and the current individual (p) are converted into binary format, and a comparison is made based on their respective i^{th} bit positions. As an illustration, suppose the best individual after a generation is represented as (4 3 3 2 3 1 2 1 4 1 2 4) and one of the current individuals is (2 1 3 3 2 1 4 3 4 4 1 2). Their corresponding binary formats are:

$$b = 10\mathbf{0}\ 011\ 011\ 010\ 011\ 001\ 010\ 001\ 100\ 001\ 010\ 100$$

$$p = 01\mathbf{0}\ 001\ 011\ 011\ 010\ 001\ 100\ 011\ 100\ 100\ 001\ 010$$

If the rotation gate is to be applied at the $3rd$ index, then we have: $i = 3$, $p_i = 0$, $b_i = 0$, $f(p) < f(b) =$ True, $\Delta\theta = 0$, and $s(\alpha_i, \beta_i) = 0$. We utilize these values to obtain new probability amplitudes for an individual, where $\theta = 0$. Consequently, a new individual is generated based on the updated probability amplitudes. This rotation operation shifts all individuals towards the direction of the best individual in that generation. The individuals are transformed into new probability amplitude values by employing the rotation gate. Once this step is completed, we repeat Steps 5 to 8 to generate the next generation of valid chromosomes in decimal format.

B. Schedule Builder

As this study employs an indirect chromosome representation for encoded schedules, the inclusion of a schedule builder module is necessary to transform the encoded chromosomes into valid schedules. Throughout the evaluation phase, the schedule builder's role is pivotal, selected based on the optimization performance criterion (minimizing the makespan). The existing literature also suggests that using a powerful schedule builder can improve the genetic minimum-makespan in JSSP [1]. This work uses a modified version of the hybrid Giffler & Thompson Algorithm [27] as the schedule builder that can generate an active schedule. An active schedule builder employs a type of local search that introduces heuristic improvements into the genetic search process [1]. The schedule is active if no *permissible left shift* is possible. The permissible left shift reduces the makespan by shifting an operation to an earlier position without causing delays for other jobs. By restricting the search space to active schedules, a substantial reduction in search complexity can be achieved while ensuring the possibility of finding an optimal schedule. This work's modified version of the algorithm integrates the original Giffler & Thompson Algorithm to discover initial schedules. Then it uses Thompson Sampling, a concept from reinforcement learning, to select the best schedules from the generated initial schedules [28].

C. Crossover Operation

In GA, searching for the optimal solution relies on creating new individuals from existing ones. The crossover process facilitates this by exchanging genetic information between parents, resulting in chromosomes that are more likely to be superior to their parents. However, not all crossover techniques are appropriate for problems like the JSSP that use permutation-based representations. Simple techniques like single-point or two-point crossover fail to maintain the order of elements in the permutation, potentially leading to infeasible solutions. Therefore, specialized crossover operators are necessary, like the generalized order crossover (GOX), which has proven effective in permutation-based representations. The GOX can preserve the order of elements in a chromosome, making it a viable option for this problem.

$$\begin{array}{ll}
\text{Parent-1:} & [\;1\;\; \mathbf{2}\;\; \mathbf{1}\;\; \mathbf{1}\;\; \mathbf{3}\;\; 2\;\; 3\;\; 3\;\; 2\;] \\
\text{Parent-2:} & [\;1\;\; 2\;\; 2\;\; 1\;\; 3\;\; 2\;\; 3\;\; 1\;\; 3\;] \\
\text{Child:} & [\;1\;\; 2\;\; 1\;\; 1\;\; 3\;\; 2\;\; 2\;\; 3\;\; 3\;]
\end{array}$$

Fig. 3: Generalized order crossover

Fig. 3 demonstrates the GOX for a 3×3 JSSP. To begin, two substrings are chosen from the parents, ensuring they contain sufficient information. This is accomplished by selecting a substring between 30% to 50% of the parent's length. Once the substrings are chosen, the elements from Parent-1 are located in Parent-2 and removed from it. Subsequently, the elements from Parent-1 are inserted into Parent-2 while maintaining their order. The entire substring is placed where the first corresponding element was found, and the remaining elements are appended to the end of Parent-2.

D. Mutation Operation

After crossover, based on probability, parents randomly undergo mutation to alter a small portion of their chromosome. It helps to avoid getting stuck in local optima by increasing population diversity. However, due to the permutation-based representation for the JSSP, a specialized mutation operator is necessary to maintain feasibility while enabling the exploration of new schedules. This involves selecting two random jobs and swapping their positions in the schedule, resulting in a minor change that increases diversity while ensuring the schedule remains valid. Fig. 4 illustrates this mutation operation for a 3×3 JSSP. Here, the initial instance of Job-2 in Parent-1 is exchanged with the second occurrence of Job-2 in Parent-2. Likewise, the first occurrence of Job-3 in Parent-2 is interchanged with the third occurrence of Job-3 in Parent-1.

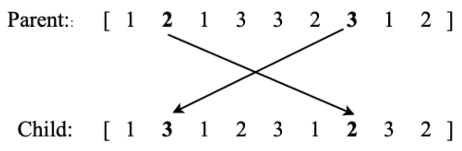

Fig. 4: The job pair exchange mutation operation.

IV. Experimental Results and Analysis

To assess the efficacy of the proposed quantum representation-based GA (QGA) and find its feasibility in addressing the JSSP, we tested it using benchmark problems in the existing literature. It includes the initial three benchmark problems (mt06, mt10, and mt20) proposed by Muth and Thompson; and the "*ten tough problems*" (abz and la problems) collected by Applegate and Cook from literature. Still, some of the "*ten tough problems*" remain unsolved. The description of the data, including problem size, best lower limit, and optimal solution status, are available in the OR-library [29]. The performance of the QGA is compared with the standard genetic algorithm (sGA) that does not utilize any quantum principles. The QGA and sGA are run for 300 iterations, the population size is 40, and crossover and mutation probabilities are 0.9 and 0.15, respectively.

TABLE III: Comparison between sGA and QGA

Instance	**Makespan (sGA)**			**Makespan (QGA)**		
	Best	**Avg**	**Worst**	**Best**	**Avg**	**Worst**
mt06	55	55	55	55	55	55
mt10	987	1012.1	1077	959	987.13	1022
abz7	735	780.6	799.2	729	767.6	782.3
abz8	756	787.33	825	747	768.16	812
abz9	771	798.2	835	759	773.01	815
la21	1081	1133.23	1170	1067	1105.56	1150
la24	1003	1023.63	1054	987	1007.63	1048
la25	1061	1092.01	1143	1047	1085.13	1121
la27	1325	1377.76	1425	1294	1331.76	1380
la29	1261	1318.66	1353	1242	1296.66	1337
la38	1323	1372.33	1426	1311	1357.23	1398
la40	1282	1368.1	1422	1261	1357.13	1407

We conducted an analysis to examine how incorporating some quantum computing benefits could enhance the algorithm. Table III shows the outcomes of the comparison. The values in the table demonstrate that using a rotation gate successfully and effectively moves all individuals towards the best individual, resulting in a lower upper bound score and, on average, better scores. The table shows that the QGA outperforms sGA in generating near-optimal schedules for all problems except mt06, where the achieved makespans are the same. The superior average values for all the benchmark problems also imply that the QGA discovers near-optimal schedules and demonstrates consistent performance across generations for most of the population. This result supports the notion that incorporating quantum phenomena enhances the GA's overall convergence. It is worth noting that in some isolated cases, the QGA produces a relatively worse outcome than the sGA when examining the worst value. This outcome could be attributed to the mismatch between the binary-to-decimal format for quantum crossover operators, as well as the absence of an elitism mechanism in the QGA framework for solving the JSSP. Nevertheless, this is a rare occurrence, and considering the QGA's better performance across the benchmark problems, its overall performance is promising and efficient.

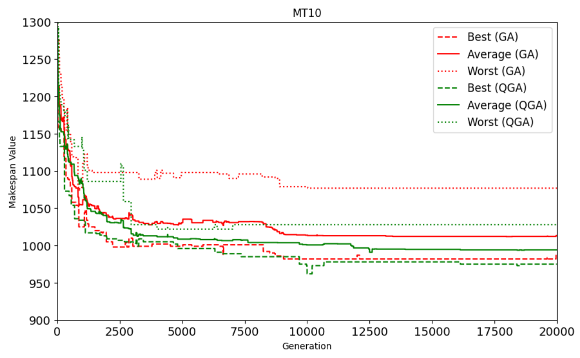

Fig. 5: Comparison of QGA and sGA on mt10.

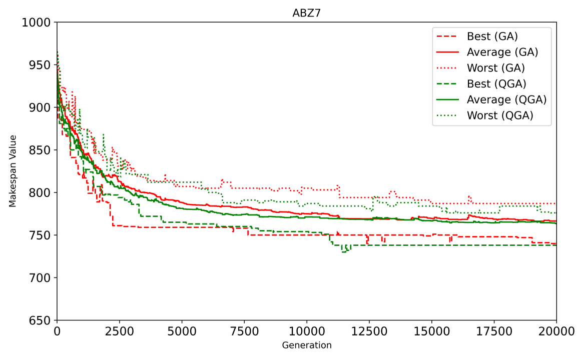

Fig. 6: Comparison of QGA and sGA on abz7.

Fig. 5 and Fig. 6 compare the performance of the QGA and sGA on the mt10 and abz7 benchmark problems, respectively. As illustrated in these figures, the QGA outperformed the sGA and exhibited smaller fluctuations in makespan. Evidently, the QGA performed significantly better for the mt10 and showed slight improvement for the abz7. These figures also visually illustrate the optimization behaviour of the solutions obtained through QGA and sGA. Both QGA and sGA begin with randomly selected initial solutions that are almost identical. However, in just a few generations, QGA demonstrates faster convergence than sGA, and this trend persists until termination. It proves the better convergence capability of the QGA.

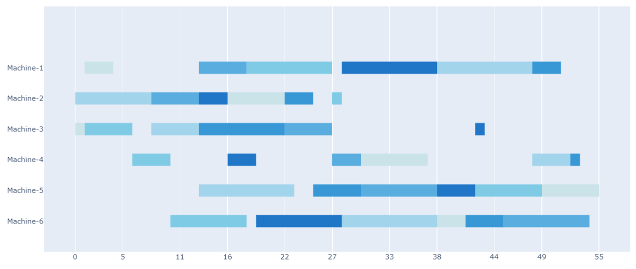

Fig. 7: Gantt chart for mt06.

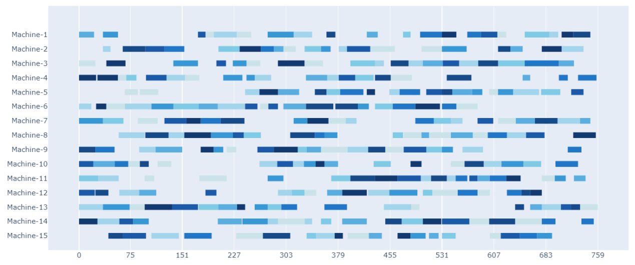

Fig. 8: Gantt chart for abz9.

Fig. 7 and Fig. 8 depict the Gantt charts, showing the makespan value, for the mt06 and abz9 problems. In summary, QGA is found to perform better than sGA for smaller benchmarks, but the difference in performance is not as significant for larger benchmarks with a wider search space. Various aspects of QGA were modified to see if performance could be improved, but some quantum crossover methods did not seem to provide any noticeable gains in performance. One possible reason for this could be that the quantum crossover operation was in binary format, while the problem required decimal notation, which may have impacted the learning process due to the need for conversion to a suitable format. An alternative representation that requires minimal changes could be utilized to address this issue. To improve performance, other aspects of the QGA could be modified, such as preventing solutions from getting stuck at local optima, using multiple starting populations, or breaking the starting population into competing subgroups to obtain the best individuals. These modifications could lead to a more diverse solution and a more optimal or near-optimal makespan.

V. Conclusion

This paper proposes a quantum representation-based genetic algorithm (QGA) to solve the job shop scheduling problem (JSSP) on classical computers. Although there has been limited research in scheduling incorporating quantum phenomena, the existing works are primarily designed for quantum processors. In contrast, this paper utilizes *qubits* as a probabilistic representation for solving the JSSP within the QGA to be executed on classical computers. The update mechanism for the QGA uses the quantum rotation gate, which helps guide the search direction towards the optimal solution. While the primary objective of this paper is to assess the applicability of the QGA in addressing JSSPs, with less emphasis on attained performance, the paper evaluates the QGA's effectiveness through experiments conducted on benchmark JSSPs of varying sizes. Additionally, it provides a comparison with the standard genetic algorithm (sGA). The results indicate its feasibility and efficiency in solving JSSPs. However, the QGA sometimes struggles with larger benchmark JSSPs due to the conversion required from binary to decimal format for quantum crossover operators. In future, it is essential to focus on designing quantum crossover operators that eliminate this conversion requirement. Another natural extension for future research is developing an elitism mechanism for the QGA.

REFERENCES

[1] K. S. N. Ripon, C.-H. Tsang, and S. Kwong, "An evolutionary approach for solving the multi-objective job-shop scheduling problem," *Evolutionary Scheduling*, pp. 165–195, 2007.

[2] K. S. N. Ripon, N. H. Siddique, and J. Torresen, "Improved precedence preservation crossover for multi-objective job shop scheduling problem," *Evolving Systems*, vol. 2, pp. 119–129, 2011.

[3] M. Karimi-Mamaghan, M. Mohammadi, P. Meyer, A. M. Karimi-Mamaghan, and E.-G. Talbi, "Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art," *European Journal of Operational Research*, vol. 296, no. 2, pp. 393–422, 2022.

[4] K. S. N. Ripon, K. Glette, M. Høvin, and J. Tørresen, "Multi-objective evolutionary approach for solving facility layout problem using local search," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, 2010, pp. 1155–1156.

[5] A. Slowik and H. Kwasnicka, "Evolutionary algorithms and their applications to engineering problems," *Neural Computing and Applications*, vol. 32, pp. 12 363–12 379, 2020.

[6] J. Holland, "Adaptation in natural and artificial systems mit press," *Cambridge, MA*, 1975.

[7] B. Akay and X. Yao, "Recent advances in evolutionary algorithms for job shop scheduling," *Automated Scheduling and Planning: From Theory to Practice*, pp. 191–224, 2013.

[8] L. Davis, "Job shop scheduling with genetic algorithms," in *Proceedings of the first International Conference on Genetic Algorithms and their Applications*. Psychology Press, 2014, pp. 136–140.

[9] R. Cheng, M. Gen, and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithms, part ii: hybrid genetic search strategies," *Computers & Industrial Engineering*, vol. 36, no. 2, pp. 343–364, 1999.

[10] J. Gu, X. Gu, and M. Gu, "A novel parallel quantum genetic algorithm for stochastic job shop scheduling," *Journal of Mathematical Analysis and Applications*, vol. 355, no. 1, pp. 63–81, 2009.

[11] A. Malossini, E. Blanzieri, and T. Calarco, "Quantum genetic optimization," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 2, pp. 231–241, 2008.

[12] M. Moore and A. Narayanan, "Quantum-inspired computing," *Dept. Comput. Sci., Univ. Exeter, Exeter, UK*, 1995.

[13] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994, pp. 124–134.

[14] L. K. Grover, "Quantum mechanics helps in searching for a needle in a haystack," *Physical review letters*, vol. 79, no. 2, p. 325, 1997.

[15] G. Zhang, "Quantum-inspired evolutionary algorithms: a survey and empirical study," *Journal of Heuristics*, vol. 17, no. 3, pp. 303–351, 2011.

[16] K.-H. Han and J.-H. Kim, "Genetic quantum algorithm and its application to combinatorial optimization problem," in *Proceedings of the 2000 congress on evolutionary computation. CEC00 (Cat. No. 00TH8512)*, vol. 2. IEEE, 2000, pp. 1354–1360.

[17] A. Narayanan and M. Moore, "Quantum-inspired genetic algorithms," in *Proceedings of IEEE international conference on evolutionary computation*. IEEE, 1996, pp. 61–66.

[18] R. Aggoune, "Quantum solutions to job shop scheduling problems," in *24ème édition du congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision ROADEF 2023*, 2023.

[19] D. Amaro, M. Rosenkranz, N. Fitzpatrick, K. Hirano, and M. Fiorentini, "A case study of variational quantum algorithms for a job shop scheduling problem," *EPJ Quantum Technology*, vol. 9, no. 1, p. 5, 2022.

[20] B. Denkena, F. Schinkel, J. Pirnay, and S. Wilmsmeier, "Quantum algorithms for process parallel flexible job shop scheduling," *CIRP Journal of Manufacturing Science and Technology*, vol. 33, pp. 100–114, 2021.

[21] J. Gu, C. Cao, B. Jiao, and X. Gu, "An improved quantum genetic algorithm for stochastic job shop problem," in *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, 2009, pp. 827–830.

[22] A. Lentzas, C. Nalmpantis, and D. Vrakas, "Hyperparameter tuning using quantum genetic algorithms," in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2019, pp. 1412–1416.

[23] H. Wang, J. Liu, J. Zhi, and C. Fu, "The improvement of quantum genetic algorithm and its application on function optimization," *Mathematical problems in engineering*, vol. 2013, 2013.

[24] K.-H. Han, K.-H. Park, C.-H. Lee, and J.-H. Kim, "Parallel quantum-inspired genetic algorithm for combinatorial optimization problem," in *Proceedings of the 2001 congress on evolutionary computation (IEEE Cat. No. 01TH8546)*, vol. 2. IEEE, 2001, pp. 1422–1429.

[25] C. Durr and P. Hoyer, "A quantum algorithm for finding the minimum," *arXiv preprint quant-ph/9607014*, 1996.

[26] H. Y. Wong, "Walsh–hadamard gate and its properties," in *Introduction to Quantum Computing: From a Layperson to a Programmer in 30 Steps*. Springer, 2022, pp. 153–162.

[27] B. Giffler and G. L. Thompson, "Algorithms for solving production-scheduling problems," *Operations research*, vol. 8, no. 4, pp. 487–503, 1960.

[28] C. Bierwirth, "A generalized permutation approach to job shop scheduling with genetic algorithms," *Operations-Research-Spektrum*, vol. 17, no. 2-3, pp. 87–92, 1995.

[29] "OR library," http://mscmga.ms.ic.ac.uk, accessed: 30-03-2023.