Journal of Combinatorial Optimization (2025) 50:33
https://doi.org/10.1007/s10878-025-01347-7

A quantum genetic algorithm for a parallel machine scheduling problem

Tilmann Schwenzow[1,3] · Annika Lehnert[2] · Christoph Liebrecht[1] · Jörg Franke[3] · Sebastian Reitelshöfer[3]

Accepted: 5 August 2025

Abstract

Scheduling problems are a common challenge across various industries. This paper addresses a specific problem arising in printed circuit board (PCB) assembly: the parallel machines scheduling problem with sequence-dependent setup times. To address this challenge, we propose a hybrid quantum genetic algorithm (QGA) designed to solve this problem on an error-free quantum computer. The development focuses on three key aspects: efficient adaptability of the quantum circuit to different problem instances, feasibility of the measured circuit outputs, and efficient utilization of qubits. To compare the quantum algorithm with its purely classical counterpart, we introduce a novel key performance indicator to quantify a potential quantum speedup. Furthermore, we evaluate the convergence behavior of the solver across various problem instances. Our results demonstrate that the QGA exhibits strong convergence behavior, resulting in near-optimal solutions. Additionally, we identify a potential quantum advantage for solving this practical problem. The advantage increases as the population size grows.

Keywords Parallel machines scheduling · Quantum genetic algorithm · Grover's algorithm · Hybrid quantum algorithm

✉ Tilmann Schwenzow
tilmann.schwenzow@faps.fau.de

[1] Manufacturing, Siemens AG, Oestliche Rheinbrueckenstr. 50, 76187 Munich, Germany

[2] Fakultaet für Mathematik, Karlsruher Institut für Technologie (KIT), Englerstraße 2, 76131 Karlsruhe, Germany

[3] Institute for Factory Automation and Production Systems (FAPS), Friedrich-Alexander-Universitaet Erlangen-Nuernberg (FAU), Egerlandstraße 7, 91058 Erlangen, Germany

Published online: 18 October 2025

1 Introduction

scheduling problems are challenges in production planning across various industries (Theradapuzha Mathew and Johansson 2023). The scheduling problem inspiring this paper originates from printed circuit board (PCB) assembly. In this context, a diverse product range is manufactured on multiple identical machines. Each job represents the production of a batch of printed circuit boards (PCBs), which can generally be produced on any of the available machines. Once a job, for example, the production of 200 PCBs, is started on a machine, it is typically completed on that machine even in the event of a disruption. This is due to the long setup times associated with jobs.

Sequence-dependent setup times occur between products belonging to different setup groups, leading to increased complexity. For instance, setup operations such as changing printing stencils can vary in duration depending on the current and next job. For instance, the stencil change between certain jobs with different setup groups can be particularly error-prone and time-consuming. The goal is to schedule the production program to minimize the makespan.

Formally, this problem can be classified as $P|ST_{sd}|C_{\max}$ according to Lawler et al. (1993), where P denotes parallel machines, ST_{sd} represents sequence-dependent setup times (Hamzadayi and Yildiz 2017), and $C_{\max}$ is the objective to minimize the makespan. The computational complexity of this problem increases rapidly with problem size, as it is classified as NP-hard (Kravchenko and Werner 1997).

While exact methods, such as Branch-and-Bound solvers, can guarantee optimal solutions, they are often computationally prohibitive for large-scale problems (Korte and Vygen 2018; Thakoor et al. 2009). Metaheuristics, such as Genetic Algorithms (GAs), first proposed by Holland in 1992 (Katoch et al. 2021), are problem-agnostic methods that typically provide good, though not always optimal, solutions. Due to their flexibility and efficiency, metaheuristics have become a standard approach for tackling NP-hard problems (Almufti et al. 2023; Karger et al. 2010).

In practice, over 150 jobs are typically scheduled on more than 10 parallel machines in PCB assembly for a single day. The creation of an optimal schedule is excluded by the nature of the NP-hard problem using classical exact methods. Non-exact methods, such as metaheuristics, usually provide solutions in a short time, but often these solutions are suboptimal

Achieving high solution quality within short computational times necessitates either the development of new algorithms or the innovative integration of existing methods. In this context, quantum computing presents a promising opportunity (Phillipson 2024). With its unique capability for quantum parallelism, quantum computing has the potential to significantly enhance computational efficiency (Nielsen and Chuang 2010). While theoretical advantages have been demonstrated—notably through Grover's (1996) and Shor's (1997) algorithms—the practical application of quantum computing to real-world problems, such as the one addressed in this paper, remains an open and exciting area of research.

1.1 Literature analysis

Existing quantum genetic algorithms (QGAs) in the current research landscape can be categorized into three types: quantum-inspired algorithms executed entirely on classical computers, hybrid algorithms combining quantum and classical hardware, and fully quantum algorithms designed for quantum hardware. An overview of the compared approaches is provided in "Appendix A".

1.1.1 Quantum-inspired GAs

The algorithm developed by Lahoz-Beltra (2016) is executed entirely on classical hardware. It translates quantum gates to classical operations, where they serve as genetic operators. Since the algorithm does not leverage quantum hardware, it does not achieve any quantum advantage. Furthermore, the handling of infeasible solutions is not discussed, making it difficult to assess its applicability to scheduling problems. In such problems, for instance, a job might be scheduled multiple times or not at all–violations that must be actively prevented.

1.1.2 Hybrid QGAs

Acampora and Vitiello (2021) propose a hybrid algorithm in which the population is represented in quantum states, although without using superposition. The fitness function is computed classically, and the results are used to guide quantum implementations of genetic operators such as crossover and mutation. However, some parameters, such as those used for mutation, are predefined at the classical level. While the algorithm is general-purpose, it does not address problem-specific constraints or the handling of invalid solutions arising from specific domains. In scheduling problems, the primary computational effort lies in evaluating the quality of individual schedules–a component where quantum speedup would be most beneficial. Since the fitness calculation is performed classically, this potential advantage is not exploited.

Malossini et al. (2008), on the other hand, introduce a hybrid algorithm where crossover and mutation are conducted classically, while selection and fitness calculation are handled quantum mechanically. Malossini et al. use the Dürr-Høyer algorithm (Durr and Hoyer 1999) to identify subpopulations with relatively high fitness rather than seeking an optimal solution in each iteration. However, like Acampora and Vitiello's work, Malossini et al.'s algorithm is not tailored to a specific problem, leaving the treatment of invalid solution spaces unaddressed.

1.1.3 Full GAs

Udrescu et al. (2006) propose a Reduced Quantum Genetic Algorithm (RQGA) in which fitness calculation is performed within a quantum circuit. However, only a single genetic algorithm iteration is conducted, and neither crossover nor mutation is employed. The best solution is identified using the Boyer-Brassard-Hoeyer-Tapp algorithm (Boyer et al. 1998). The approach does not include mechanisms to avoid infeasible solutions, such as jobs being scheduled more than once or not at all. This

omission increases the risk of the optimization process moving through an invalid solution space.

In subsequent work, Ardelean and Udrescu (2024) adapt this approach by fixing certain qubits to classical values, thereby reducing the search space and improving efficiency. Although this technique does not eliminate the issue of infeasible solutions, it significantly reduces its impact. As the approach involves only a single step of the genetic process, crossover and mutation are not performed.

SaiToh et al. (2014) introduce a QGA incorporating both crossover and mutation. The Dürr-Høyer algorithm is used to identify the best solution. However, some components of the algorithm rely on pseudo-random values generated classically, which undermines the algorithm's ability to freely explore the solution space through purely quantum means. Infeasible solutions are not addressed, as the algorithm is not designed for a specific application. When adapting the algorithm to scheduling problems, the solution space must be carefully restricted to ensure feasibility.

The analysis of existing QGA approaches highlights a lack of practical implementations for quantum fitness calculations. Most algorithms are general-purpose and not tailored to specific applications, such as scheduling. Scheduling problems introduce constraints, as described in Sect. 1, which can result in infeasible solutions when quantum superpositions are used. One possible solution to address this issue is the introduction of penalties for constraint violations. However, as the population size or solution space grows (e.g., with a larger number of individuals per generation), the proportion of infeasible solutions often grows disproportionately, threatening the long-term efficiency of the algorithm.

This paper aims to address these research gaps by proposing a QGA specifically designed for scheduling problems. The algorithm will be evaluated for its practical applicability and effectiveness in solving such problems, with particular attention to the prevention of infeasible solutions. We design a QGA in such a way that the central optimization component, the fitness calculation, is executed quantum-mechanically to enable potential quantum speedup, while simultaneously ensuring that the search is confined to the legal solution space through algorithmic design. The key objective is to assess whether such an algorithm can achieve a measurable quantum advantage.

2 Quantum genetic algorithm design

Quantum simulators on standard hardware are generally limited to simulations of up to 30 qubits, and real quantum computers are not yet error-free enough for the useful application of Grover's algorithm. This paper designs an operational quantum circuit within these constraints. "Operational" here means that small scheduling examples (e.g., 3 jobs and 2 machines) can be solved. Until today, it is not possible to solve a practical problem with over 150 jobs and multiple machines with a QGA on real hardware. Nevertheless, in future more robust quantum systems could enable the solution for industrial-scale problems, including those found in PCB assembly. In anticipation of this progress, the quantum circuit presented in this paper is designed to be general, so that the quantum code does not need to be adapted to different problem instances.

Figure 1 shows the basic structure of the developed hybrid quantum genetic algorithm (QGA), distinguishing between classical and quantum components. This distinction was chosen for specific reasons: selection and fitness calculation are implemented quantum mechanically to achieve a quantum advantage in finding the best schedule. Mutation is also implemented quantum mechanically, as randomness is well-represented in quantum computing and mutation can be controlled to ensure feasible solutions. Classical components, such as the creation of the initial population and crossover, ensure feasibility through validation routines, which are easier to implement classically. Extending these components into the quantum domain is a potential area for future research.

The following sections describe each component of the algorithm in detail: Encoding, as the foundation of the algorithm, is implicitly included and supports the following components (Fig. 1): Creation of Initial Population, Crossover, Mutation, Fitness Evaluation, Selection, and Termination Criterion.

2.1 Encoding

Scheduling information is encoded in two parts: the global sequencing of jobs and their allocation to machines (see Fig. 2). For clarity, we refer to the example in Fig. 2, focusing on individual a). In this example, job 1 (binary 01) is at position 0, job 0 is at position 1, and job 2 is at position 2. Jobs 1 and 0 are assigned to machine 0, while job 2 is assigned to machine 1.

This encoding is instantiated in qubits using binary representation. To enable superposition and exploit the quantum advantage, address qubits are added to represent different individuals. Experiment-specific data, such as setup groups, initial machine setups, setup durations, and job durations, are also initialized in the quantum

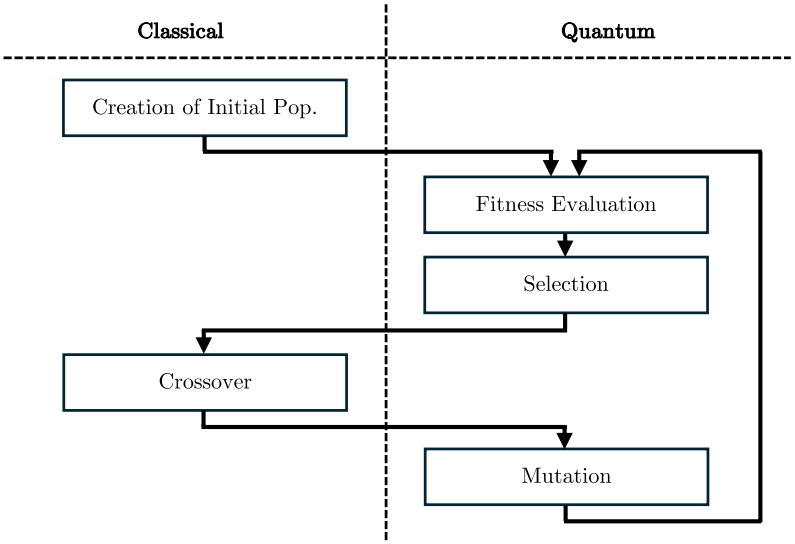

Fig. 1 Structure of the developed hybrid Quantum Genetic Algorithm (QGA)

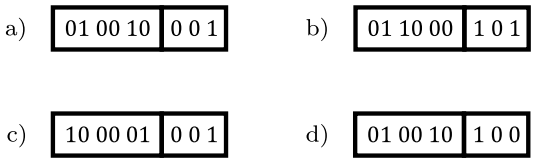

Fig. 2 Population of 4 individuals. **a** First job 1, then job 0, then job 2. Jobs 1 and 0 on machine 0, job 2 on machine 1. **b** First job 1, then job 2, then job 0. Job 2 on machine 0, jobs 0 and 1 on machine 1. **c** First job 2, then job 0, then job 1. Jobs 2 and 0 on machine 0, job 1 on machine 1. **d** First job 1, then job 0, then job 2. Job 1 on machine 1, jobs 0 and 2 on machine 0

register. To save qubits, reusable values are shared wherever possible. Additional ancilla qubits and result qubits are included to facilitate operations and store the measured makespan.

2.2 Creation of initial population

A population of 2^n individuals is generated classically, where n is a natural number, ensuring all solutions are feasible. This size is chosen to fully utilize the address qubits. After transferring the population to the quantum computer, the address qubits are put into superposition of all individuals. The creation of the population is based on (Malossini et al. 2008).

2.3 Fitness calculation

The makespan represents the fitness of a schedule. The goal is to minimize the makespan. The makespan is calculated quantum mechanically by using three quantum techniques: Firstly, the Quantum Fourier Transform Adder (QFT Adder) by Draper (2000), which adds integer numbers in the quantum circuit. Secondly, the parallel integer comparator by Thula (2023), which compares two integer numbers in the quantum circuit. Thirdly, the integer comparator by IBM (2024), which compares a classical integer number and an integer number in the quantum circuit. Superposition enables simultaneous fitness calculation for all individuals. The calculation is divided into four components: initial setup change, setup changes between jobs, job durations, and machine comparison.

2.4 Initial setup change

Figure 3 illustrates the addition of initial setup times for the first job on a machine. The example of Fig. 2a with job 1 is at position 0 on machine 0 is illustrated in Fig. 3. In this Example, all control qubits are 1 for the second QFT Adder. Therefore the QFT Adder is activated, and the setup change is added to the makespan for machine 0. X-gates flip control ancillae to prevent redundant additions after the QFT Adders. This process is repeated for all machines.

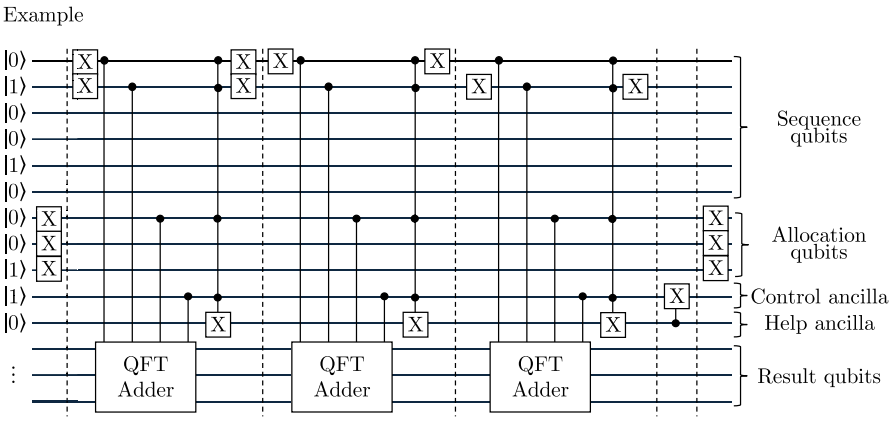

Fig. 3 Circuit for the initial setup change for the job in position 0

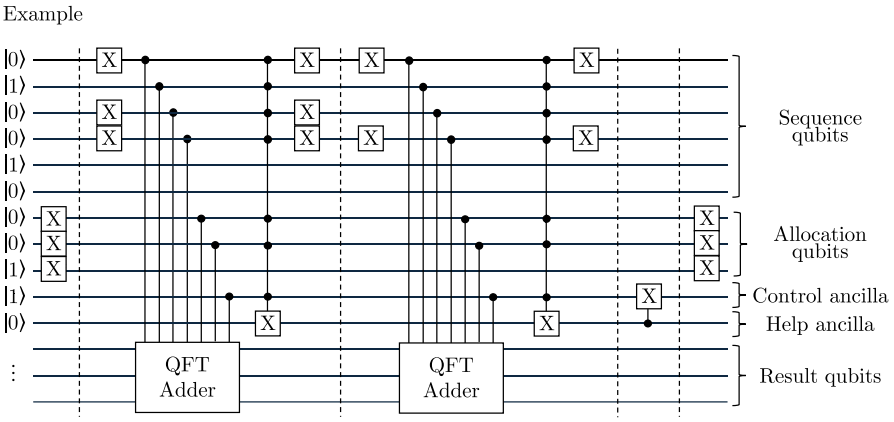

Fig. 4 Circuit for adding the setup change between job 1 at position 0 and the job at position 1

2.5 Setup changes between jobs

Figure 4 shows the addition of setup changes between jobs at consecutive positions. For example, if job 1 is at position 0 and job 0 is at position 1, the control qubits are flipped to 1. When all control qubits are 1 an addition is triggered, and subsequent redundancies are prevented. All job pairs and positions are processed in the same way.

2.6 Job durations

Figure 5 shows the addition of job durations. Control qubits are set to 1, when the necessary job is on the certain machine. When all control qubits are 1, the QFT Adder is triggered. For example, in Fig. 4, job 0 is added at position 1 on machine 0. This process is repeated for all jobs and machines.

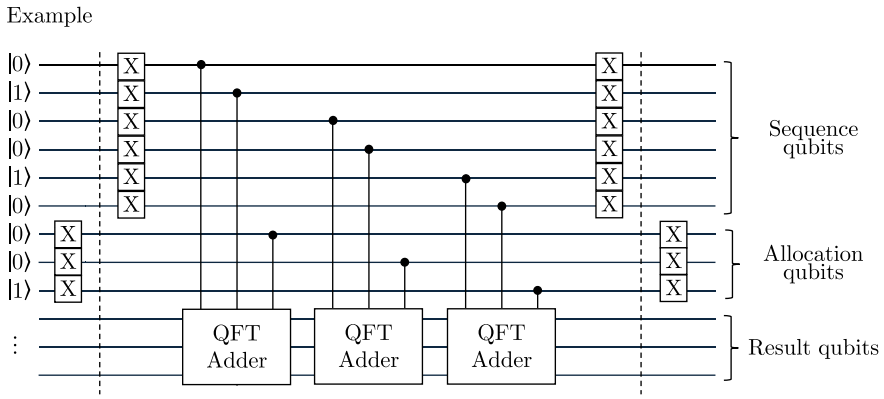

Fig. 5 Circuit for adding the duration of job 0 on machine 0

2.7 Machine comparison

After calculating intermediate sums for each machine, the parallel integer comparator identifies the largest value, representing the makespan and the individual's fitness and changes the result qubits to this value.

2.8 Selection

Selection aims to identify the individual with the smallest makespan. Direct measurement after fitness calculation would collapse the superposition and provide equal probabilities for all individuals. To address this, a variant of the Dürr-Høyer algorithm is used to increase the probability of measuring the best solution.

First, a measurement of the fitness with no Grover iteration is executed to establish an initial threshold. For a population size of N individuals, the number of Grover steps is incrementally increased in each iteration, starting from 0 up to $\left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$. After the number of Grover steps, there is always a calculation of the makespan resulting in a measurement of it. During each iteration, the algorithm searches for a solution with a makespan smaller than the current threshold. If such a value is found, the threshold is updated, and the iteration concludes.

This iterative process continues until a predefined number of iterations without further improvement is reached, ensuring that the best solution is identified with high probability. A pseudocode of the selection process is in "Appendix B" Algorithm 1.

2.9 Crossover

After completing the selection process to identify the best individual within the population, the crossover operation is performed on classical hardware. Since the best individual of the population is known after the selection step, crossover is applied between this best individual and other individuals in the population. This approach

integrates promising solution components from the best individual into others, aiming to generate high-quality offspring.

In each iteration, crossover is applied to approximately $round(pop\_size/1.3)$ individuals. The factor 1.3 was identified as effective in a preliminary study based on 100 test instances. It ensures a large proportion of individuals undergo crossover while leaving a small fraction to be replaced by entirely new, randomly generated individuals. This hybrid approach introduces fresh genetic material into the process, enhancing exploration while maintaining the integrity of high-quality solutions. While the chosen factor proved suitable for the test cases used, its effectiveness may vary depending on the specific problem structure. Tuning this parameter for other applications could further improve performance.

The selection of individuals for crossover follows a random principle, excluding the best individual to ensure its preservation in following generations. There are three crossover methods of choosing which part of the individual is the crossover applied to. Choosing the sequence part, the allocation part, or both simultaneously. One of the three methods is chosen randomly. To further increase variation, the number of crossover points is varied between 1 and $round(1/3 \cdot num\_jobs)$. This randomized approach ensures diversity within the population while maintaining feasibility through repair mechanisms.

The *n*-point crossover guarantees that the allocation part of the solution remains feasible after the operation. However, for the sequence part, *n*-point crossover can result in infeasible solutions. To address this, a repair mechanism is implemented, ensuring that the sequence remains valid after crossover.

Once the population resulting from crossover is prepared, it is transferred back to the quantum circuit. During this transfer, the best solution is assigned to address qubit 00.

2.10 Mutation

In the current implementation, mutation is exclusively applied to the sequence part of the solutions. Preliminary investigations have shown that this focus yields the most significant improvements in solution quality. While extending mutation to the allocation part is conceptually possible, this remains a topic for future research.

Currently, a swap mutation is implemented for the sequence part. The mutation process introduces randomness through the use of ancilla qubits in superposition states. Two types of ancilla qubits are employed: address ancilla qubits and job ancilla qubits.

The address ancilla qubits determine which individuals in the population undergo mutation. To ensure the best individual remains unaffected, mutations are only applied to individuals whose address qubits are not all zeros (the address of the best individual). To regulate the number of mutations, only approximately $round(num\_address\_qubits \cdot 9/10)$ address ancillae are utilized. This approach increases the likelihood of multiple mutations in larger populations. For example, with a single address ancilla, mutations would occur for both individuals at addresses 10 and 11 when the ancilla collapses to 1. The job ancilla qubits determine which jobs within a sequence are swapped. A swap occurs only when exactly two of the job ancillae collapse to a value of 1. The two corresponding jobs in the sequence are then exchanged.

If fewer or more than two job ancillae have the value 1, no swap occurs. A potential limitation arises with an increasing number of jobs, as the probability of exactly two job ancillae being 1 decreases. This could be mitigated by reducing the number of address ancillae. An example circuit for mutation can be found in "Appendix C" Fig.9.

2.11 Termination criterion

The algorithm terminates after a predefined number of generations. This straightforward criterion ensures a fixed computational budget and allows for consistent comparisons across different problem instances and parameter settings. For this paper, the number of generations is set to 50.

The complete QGA is implemented in Python version 3.12.4. The quantum algorithms are developed using the Qiskit library version 1.2.0 and the Qiskit Aer package version 0.14.2.

3 Results

The results are presented in two parts. First, the focus is on evaluating a potential quantum advantage, or disadvantage, in terms of a faster selection process (Sect. 3.1). For this, only the quantum search process is compared to its classical counterpart.

In the second part, the overall effectiveness of the quantum genetic algorithm (QGA) in finding optimal solutions is assessed (Sect. 3.2).

All experiments are performed on a CPU-based system with the following specifications: a 6-core processor with 12 logical processors (Intel(R) Core(TM) i7-9850 H CPU @ 2.60GHz) and 16GB of RAM.

3.1 Discussion of quantum advantage

The goal of the search algorithm developed in Sect. 2 is to identify the individual with the best makespan within a population. To evaluate the effectiveness of the algorithm, a comparison with a classical search algorithm is conducted. In a classical search the makespan for each schedule is calculated to determine the optimal schedule.

A direct comparison between QGA and its classical counterpart is challenging because the quantum circuit, due to the error-prone nature of quantum hardware, is simulated rather than executed on real quantum devices. As such, runtime performance on actual hardware cannot be directly assessed. Instead, we develop a key performance indicator (KPI) to compare the quantum and classical algorithms for this study.

This KPI represents the quantum advantage. Simply explained, the measure developed in this paper estimates how much faster the quantum search process is compared to a purely classical search. The computation incorporates both the expected runtime until a solution is found and the probability of successfully finding that solution.

To illustrate this concept, consider the analogy of a dice game: imagine a die with differently sized faces resting on a table with the number 1 facing up. Our goal is to obtain a 6. In the quantum approach, we simulate repeatedly rolling the die until a 6 appears. Each roll represents a quantum search step. In contrast, the classical

approach would involve turning the die from one face to the next, in a fixed order, until the 6 is found. In both cases, we can count the steps taken to reach the solution. At the end of the experiment, we compare which method reached the goal faster on average.

In quantum computing, the advantage is typically evaluated through query complexity. Query complexity is a computational model that measures algorithmic efficiency based on the number of queries required to compute a function $f(x_1, \ldots, x_N)$ (Ambainis 2017). A logical step in the code is defined as a "step," and the number of steps required to solve the problem determines the query complexity. For the scheduling problem at hand, a Grover iteration is defined as one "step," and the subsequent calculation of the makespan constitutes another "step." For example, one Grover iteration followed by a measurement accounts for two steps.

This definition aligns with the classical algorithm, as a lookup of the makespan in the classical counterpart is also counted as one step. This ensures a fair comparison between the classical search algorithm and the Grover-based search algorithm.

To analyze the quantum algorithm for a given population k, the process is modeled as a probability tree (see Fig. 6). Each path in the tree represents a possible course the algorithm can take. In the figure, the calculation of the makespan is depicted as  , and one Grover iteration, consisting of the Oracle and Diffusion Operator (OD), is depicted as  .

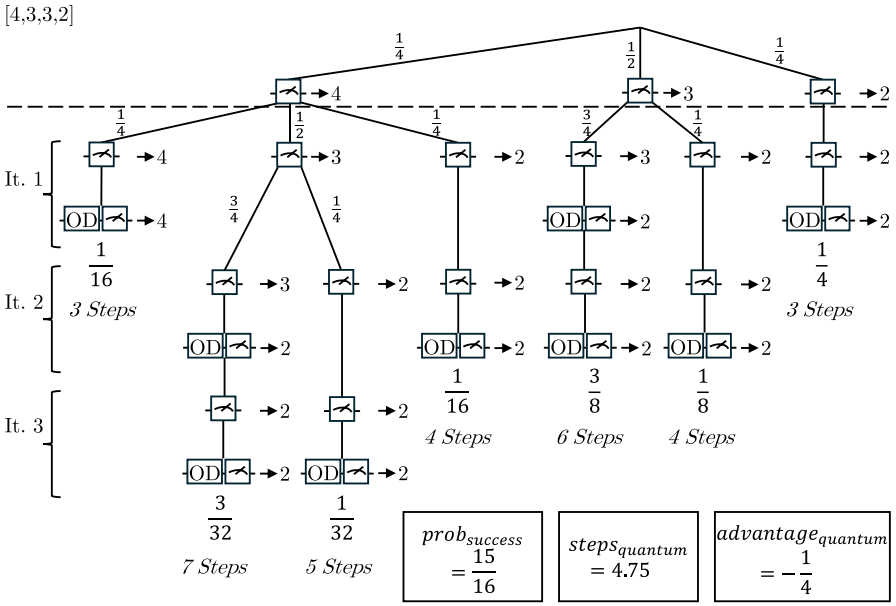

Fig. 6 Example tree for the quantum search algorithm with an example of a population with the size of 4. [4,3,3,2] represent the makespans of the four schedules. Each branch represents a possible measurement outcome after a given number of Grover iterations. Each iteration is represented by a curly bracket on the left side. The Grover iterations which are performed before the measurement are expressed by OD. On the right side of the depicted symbols are the found values. At the bottom of a path, there is the number of quantum steps and the probability of the branch

The first level of the tree represents the measurement used to determine the initial threshold, based on the structure of the given initial population. This step is excluded from the step calculations as it only occurs in the first generation. In subsequent generations, the threshold is set by the best value of the previous generation, eliminating the need for further initial measurements. To ensure consistency across generations, the step count begins only after the threshold is determined.

In the probability tree, P_k represents the set of all possible paths, and p_i is the probability associated with the i-th path. p_i can be calculated using Formula 1, originally introduced by Grover (1996): The formula calculates the probability P of finding a marked value when t out of N entries are marked, and j Grover iterations are performed:

$$P = \sin\left((2j+1)\cdot\theta\right)^2, \tag{1}$$

where

$$\theta = \arcsin\left(\sqrt{\frac{t}{N}}\right). \tag{2}$$

While Eq. (1) determines the likelihood of a branch, the quantum advantage also depends on the steps taken along that branch by the QGA. Those steps for the population can be calculated as:

$$steps^k_{quantum} = \sum_{i \in P_k} p_i \cdot steps_i. \tag{3}$$

In classical approaches, the number of steps required $steps^k_{classical}$, corresponds to the size of the respective population. This reflects the fact that, in classical approaches, all entries must be evaluated to identify the minimum value.

After calculating the steps and probabilities for each path, it is essential to assess the algorithm's effectiveness in finding the global minimum. The minimum value identified in the i-th path is denoted by $\min(i)$, while $\min_k$ represents the minimum across the entire population k. The probability of successfully identifying this global minimum is given by:

$$prob^k_{success} = 1 - \sum_{i \in P_k : \min(i) \neq \min_k} p_i. \tag{4}$$

The quantum advantage, $advantage^k_{quantum}$, is defined such that, for paths that find the minimum makespan (e.g., the right branch in Fig. 6), the relative step difference between the quantum path and the classical algorithm is considered. If the quantum path requires more steps than the classical algorithm, the term contributes negatively. For paths that do not find the minimum makespan, their total number of steps is subtracted, as every step in such paths represents a disadvantage. The paths are weighted

by their respective probabilities. To make $advantage^k_{quantum}$ comparable across different population sizes, the result is normalized by dividing it by the number of classical steps:

$$advantage^k_{quantum} = \frac{\sum_{i \in P_k,\, \min(i)=\min_k} p_i \cdot \left(steps^k_{classical} - steps_i\right)}{steps^{(k)}_{classical}} - \frac{\sum_{i \in P_k,\, \min(i)\neq\min_k} p_i \cdot steps_i}{steps^{(k)}_{classical}}. \tag{5}$$

3.1.1 Experimental planning quantum advantage

The focus of this study is to investigate a potential quantum advantage in terms of query complexity compared to a classical search algorithm. To achieve this, explicit execution of the quantum circuit is avoided due to its computational cost. Instead, the probabilities calculated using Eq. 1 are employed to simulate the behavior of the quantum circuit efficiently. This approach accurately replicates the quantum circuit's performance while significantly reducing computation time compared to direct simulation.

To isolate the search process, example populations are systematically generated, where makespan values are assigned to individuals, as illustrated in Fig. 6. These makespan values are chosen from eight distinct values ranging between one and eight time units.

For each population size N, ranging from 1 to 18, all possible combinations of the distinct makespan values are generated. While theoretical quantum circuits only make sense for population sizes that are powers of two ($N = 2^n$), to align with qubit superposition, this restriction is relaxed for the purpose of visualization. Specifically, population sizes such as $N = 3$ are included in the theoretical calculations of the quantum advantage, as the use of Eq. 1 remains valid for any N.

After analyzing the search process in the first generation of the genetic algorithm, the identified best solutions are used as thresholds for the second generation. Based on these thresholds, all subsequent combinations are generated following the same principles applied in the first generation. Therefore, the second generation extends the tree structure from the first generation, ensuring that the quantum advantage calculated for the second generation inherently accounts for the quantum advantage achieved in the first generation. In this experimental setup, crossover and mutation are intentionally omitted to focus purely on the quantum search process and its advantage.

3.1.2 Results quantum advantage

Figure 7 shows the evolution of the quantum advantage, calculated using Eq. 5, averaged over the different examples for varying population sizes. This quantum advantage compares the theoretical runtimes of the quantum search algorithm with its classical counterpart, as detailed in Sect. 3.1.

It can be observed that the overall trend indicates an improvement in the average quantum advantage, despite dips at $N = 7$ and $N = 15$. These dips occur because these population sizes correspond to increases in the maximum number of Grover iterations, $\left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$. For example, for $N = 7$.

$$\left\lfloor \frac{\pi}{4}\sqrt{7} \right\rfloor = \lfloor 2.078 \rfloor = 2,$$

whereas for $N = 6$, this number was still 1. The increase in maximum Grover steps leads to more steps per iteration, resulting in higher $steps_{quantum}$ values, which diminishes the quantum advantage and causes the respective dips in Fig. 7.

Overall, the second generation of the search algorithm shows a better quantum advantage, which was expected since the results from the first generation are used as a threshold for the second generation. This makes the search algorithm more efficient. The computational effort increases with growing population size and generation count, which is why calculations for the second generation are limited to a population size of up to 7.

3.2 Evaluation of the convergence of the QGA

The convergence of the QGA is evaluated by applying the algorithm to various scheduling instances. After the last generation of the genetic algorithm (see Sect. 2 Termination Criterion), the best makespan identified by the algorithm is recorded as the result. To allow for better comparability across schedules with different optimal makespans, the makespan found by the QGA is normalized relative to the optimal makespan. The optimal makespan is determined beforehand using an exact branch-and-bound method.

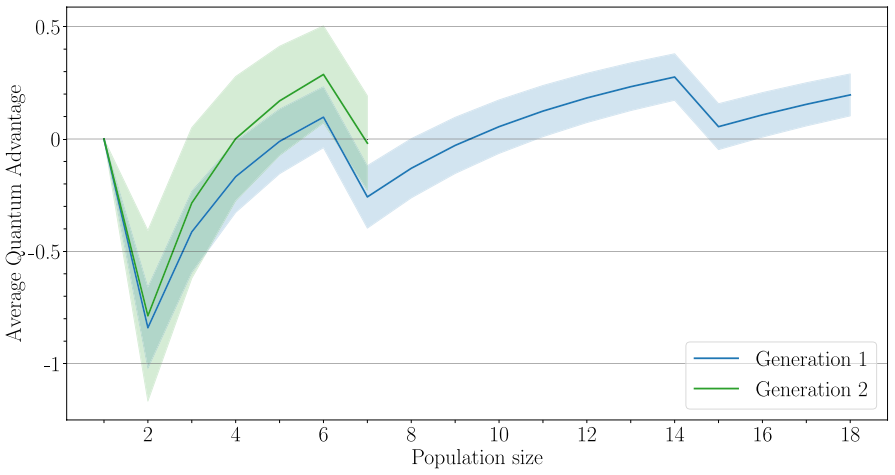

Fig. 7 Comparison of the average quantum advantage for generation 1 and 2 for all examples with different population sizes. Generations represent the generations of the genetic algorithm. The lines represents the average, while the shaded areas represent standard deviation

To assess the impact of quantum principles on solution quality, the QGA is compared to its classical counterpart. In the classical genetic algorithm, selection and mutation are implemented using classical principles rather than quantum methods. This comparison highlights the specific contributions of the quantum components.

3.2.1 Experimental planning convergence of the QGA

For the experiments, scheduling instances are generated, each consisting of seven jobs, three machines, and three setup groups. Job durations are randomly distributed between four and eight time units, while setup change times are randomly distributed between one and three time units. Additionally, the initial setup of each machine is randomly assigned to one of the three setup groups. A total of 200 random instances are created for this study.

Three different population sizes (N = 4, N = 8, N = 16) are tested. The parameter for the termination criterion of the QGA is set to 50 generations.

Due to the limitation of available qubits (approximately less than 30 on used hardware), directly simulating the quantum circuit, including the quantum search algorithm, is computationally infeasible for larger instances and large population sizes. Instead, a tree-based approach is used to emulate the behavior of the quantum circuit efficiently (see Fig. 6). This approach allows us to probabilistically determine the outcomes of the quantum search process. This means, that the execution of the quantum search algorithm is simulated by sampling paths in the probability tree, like rolling dice with differently sized faces to select a branch and follow its outcome. The same principle is applied to simulate the mutation process.

All classical components of the QGA (as shown in Fig. 1) are executed without modifications. The results obtained through this approach accurately reflect the behavior of the QGA, as it would perform on an ideal, error-free quantum computer, ensuring meaningful and reliable findings.

The classical counterpart (CGA) follows the same structure as the QGA but replaces quantum operations with classical equivalents. Instead of Grover-based quantum search, the classical algorithm calculates the makespan for every individual in the population and selects the one with the best makespan. The classical mutation process mimics the quantum mutation logic described in Sect. 2, with a classical implementation that preserves feasibility. By maintaining the same underlying logic for both algorithms, this setup enables a direct and fair comparison of the QGA with its classical counterpart.

The CGA serves as a baseline in terms of solution quality, as it computes the fitness of every schedule in the population. Consequently, the QGA can only achieve solutions that are equally good or worse.

3.2.2 Results of the QGA

Figure 8 illustrates the convergence behavior of both QGA and CGA. Both algorithms show consistent improvement in solution quality over successive generations for all tested population sizes. The mean normalized makespan steadily approaches the optimal value, while the variance of the results decreases.

As the population size increases, the mean normalized makespan for both the QGA and CGA converges closer to the absolute best makespan. This is expected since larger populations have a higher probability of including the optimal schedule.

The comparison between the QGA and CGA reveals that, during the first generations of the algorithm, the CGA slightly outperforms the QGA in terms of both the mean normalized makespan and variance. This can be attributed to the inherent randomness in the QGA's selection process. Identifying the best makespan during the quantum selection relies on probabilistic outcomes. It is not guaranteed that the quantum search will always find the best solution. This strongly depends on the ratio between the number of marked solutions and the total number of solutions. If this ratio is unfavorable, the QGA may not find the optimal path. In contrast, the CGA deterministically evaluates all schedules and always selects the best one in each generation.

However, as the algorithm progresses, the QGA becomes increasingly effective. This improvement arises from the threshold enhancement observed across generations and the fact that the best solution is preserved during both mutation and crossover. As a result, the influence of occasional suboptimal outcomes from the quantum selection process diminishes over time.

Toward the final generations, the performance difference between QGA and CGA becomes negligible, with the QGA achieving nearly identical solution quality. Since the CGA serves as a baseline, this result confirms that no real quantum disadvantage

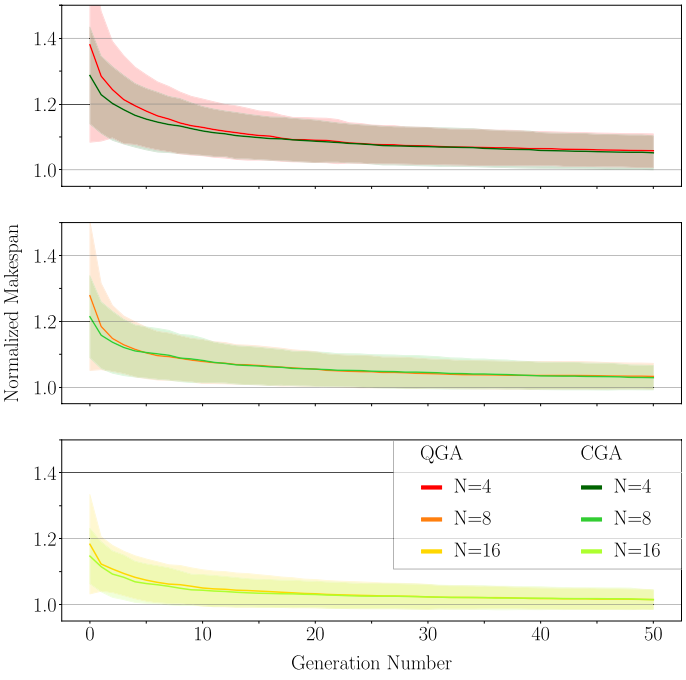

Fig. 8 Convergence of the classical (CGA) and quantum genetic algorithm (QGA) for instances with three setups, three machines, seven jobs. Lines show the average over all examples and the shaded areas represent the Standard Deviation. The subplots illustrate the results of different population sizes

in terms of solution quality can be observed. Consequently, the verified quantum advantage in terms of reduced computational steps (see Fig. 7) becomes the decisive criterion in comparing the two approaches.

4 Conclusion and outlook

The goal of this paper was to design a quantum genetic algorithm (QGA) for optimizing a parallel machine scheduling problem with sequence-dependent setup times, inspired by a practical example from PCB assembly. The objective was to minimize the makespan. The developed hybrid algorithm combines classical components of a genetic algorithm with quantum components specifically implemented for quantum computing.

We focused on implementing the evaluation and selection processes within the quantum circuit to achieve a potential quantum advantage, as selection directly addresses the core challenges of scheduling problems. For this purpose, a Grover-based approach was developed, which enhances the probability of identifying the best schedule from a population while enabling parallel makespan calculations across the entire population. Additionally, a quantum implementation of the mutation process was proposed.

Several key aspects were considered during the development of the QGA. Firstly, the quantum code was designed to be flexible, ensuring that minor modifications, such as changes to input sizes (e.g., job durations), do not require adjustments to the quantum circuit after initialization. Secondly, the number of qubits required was minimized to ensure the possibility of quantum circuit simulation. Thirdly, the feasibility of solutions was primarily maintained through classical population initialization and classical crossover operations.

To evaluate the QGA, two key experiments were conducted: Firstly, the selection process was analyzed for a potential quantum advantage, in terms of quantum speedup on an ideal quantum computer. A novel KPI, defined as the average quantum advantage, was developed. The results demonstrate a clear trend: the quantum advantage increases with population size, indicating that a quantum speedup is achievable. The advantage further improves with subsequent generations of the genetic algorithm.

Secondly, the overall performance of the QGA was evaluated by applying it to various scheduling instances. Over successive generations, the QGA steadily improves its solutions, yielding schedules with lower makespans and reducing the variance of outcomes.

When comparing the QGA to its classical counterpart (CGA)–which evaluates every schedule in the population and thus reliably identifies the best solution in each iteration–it can be observed that the QGA's performance ultimately aligns with that of the CGA. Accordingly, we see that the QGA can yield a quantum speedup without sacrificing solution quality, as its makespans of the final generation match those found by the CGA.

The promising results of this study open several avenues for future research. A key direction would be to integrate the crossover operation into the quantum circuit and investigate its effect on the quantum advantage. As quantum hardware continues to

improve and becomes less error-prone, a re-evaluation of the QGA on real quantum computers will be essential. Although the current qubit limitations prevent practical advantages for real-world scheduling problems, the roadmap of quantum computing manufacturers clearly indicates a significant increase in the number of available qubits in the coming years. The findings of this research provide a strong foundation so that hybrid optimizers may one day offer benefits in practice.

Appendix A: Literature overview and own approach

See Tables 1 and 2

Table 1 Structured comparison of Quantum Genetic Algorithms (Part 1)

Criterion	Lahoz-Beltra et al.	Acampora et al.	Malossini et al.	Udrescu et al.
Degree of quantumness	Quantum-inspired (classical)	Hybrid	Hybrid	Fully quantum
Generations	Multiple	Multiple	Multiple	One
Population representation	Classical simulation of qubit probabilities	Quantum states, no superposition	Quantum superposition	Quantum superposition
Fitness calculation	Classical	Classical	Quantum	Quantum
Mutation	Quantum-inspired	Quantum	Classical	None
Crossover	Quantum-inspired	Quantum (entanglement-based)	Classical	None
Selection	Quantum-inspired (rotation)	Classical	Quantum (Dürr-Høyer)	Quantum (Grover BBHT)
Adaptability for Scheduling Problems	Not explicitly addressed; infeasible solutions possible	Not explicitly addressed; infeasible solutions possible	Not explicitly addressed; infeasible solutions possible	Not explicitly addressed; infeasible solutions possible

Table 2 Structured comparison of Quantum Genetic Algorithms (Part 2)

Criterion	Ardelean & Udrescu	Saitoh et al.	This work
Degree of quantumness	Fully quantum	Fully quantum	Hybrid
Generations	One	Multiple	Multiple
Population representation	Quantum superposition with fixed qubits	Quantum superposition	Quantum superposition
Fitness calculation	Quantum	Quantum	Quantum
Mutation	None	Quantum	Quantum (preserves best)
Crossover	None	Quantum	Classical
Selection	Quantum (Grover BBHT)	Quantum (Dürr-Høyer)	Quantum (Grover-based threshold)
Adaptability for Scheduling Problems	Partially addressed (fixed qubits to reduce infeasible solutions)	Not addressed; infeasible solutions possible	Scheduling-specific constraints actively addressed; infeasible solutions are avoided

Appendix B: Pseudocode selection

Input: current_best, current_best_value ▷ The current best individual and value found so far
Output: current_best, current_best_value, current_best_index ▷ The minimal value such as the best individual and its index
best_ind_last_iteration ← current_best
repetition_counter ← 0
break_after ← 1 ▷ Termination criterion
while True **do**
    $m \leftarrow 0$
    max_grover_steps ← $\left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$
    while $m \leq$ max_grover_steps **do**
        Perform m **Grover steps**
        Perform measurement on the qubits
        if measurement result < current_best_value **then**
            Update current_best_index, current_best, current_best_value
            break ▷ Exit inner loop if a better solution is found
        end if
        $m \leftarrow m + 1$ ▷ Increment Grover step count
    end while
    if best_ind_last_iteration = current_best **then**
        repetition_counter ← repetition_counter + 1
    end if
    if repetition_counter =break_after **then**
        break ▷ Exit outer loop if same value measured twice
    end if
    best_ind_last_iteration ← current_best
end while
return (current_best_index, current_best, current_best_value)

Algorithm 1 Grover based search to find the best individual of a population

Appendix C: Quantum circuit

See Fig. 9.

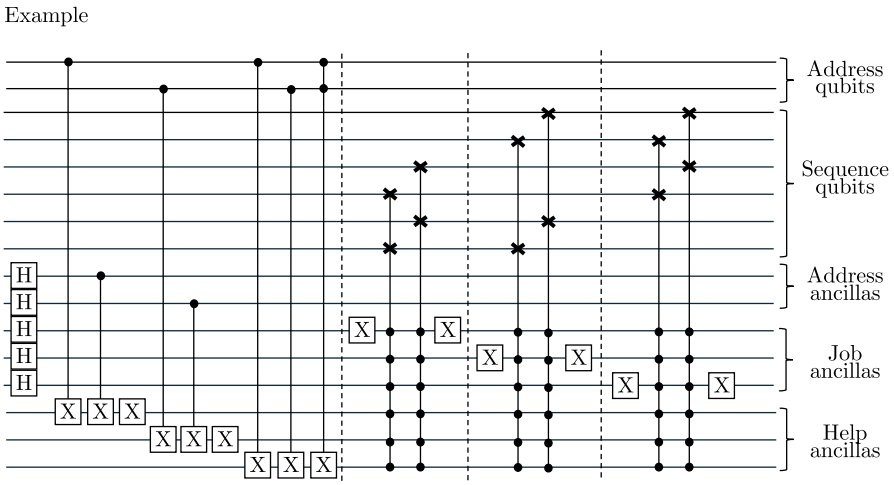

Fig. 9 Mutation circuit—the left part of the circuit shows the superposition of address- and job-ancillae and the control qubits to check for an individual, which is not the best individual (00). If all control qubits are 1 and exactly two job ancillae are 1, the swap gates are activated for the two jobs (right part)

Acknowledgements This study was done within the project "Tailored Application of Quantum Optimization for Planning and Control of Assembly and Manufacturing" (TAQO-PAM), which is funded by the Bundesministerium fuer Bildung und Forschung with the contract number 13N16093.

Author contributions AL and TS contributed to the study concept and the experimental design. AL developed the quantum circuit and the figures. TS wrote the first draft of the manuscript. All authors revised the manuscript and accepted the final manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL. This study was done within the project "Tailored Application of Quantum Optimization for Planning and Control of Assembly and Manufacturing" (TAQO-PAM), which is funded by the Bundesministerium fuer Bildung und Forschung with the contract number 13N16093.

Data availability The datasets generated during and/or analysed during the current study are not publicly available but are available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

Acampora G, Vitiello A (2021) Implementing evolutionary optimization on actual quantum processors. Inf Sci 575:542–562. https://doi.org/10.1016/j.ins.2021.06.049 (https://linkinghub.elsevier.com/retrieve/pii/S002002552100640X)

Ambainis A (2017) Understanding quantum algorithms via query complexity. https://doi.org/10.48550/arXiv.1712.06349

Ardelean SM, Udrescu M (2024) Hybrid quantum search with genetic algorithm optimization. PeerJ Computer Sci 10:e2210. https://doi.org/10.7717/peerj-cs.2210 (https://peerj.com/articles/cs-2210D)

Boyer M, Brassard G, Hoeyer P, Tapp A (1998) Tight bounds on quantum searching. Fortschr Phys 46(4):493–505. https://doi.org/10.1002/(SICI)1521-3978(199806)46:4/5<493::AID-PROP493>3.0.CO;2-P. arXiv:quant-ph/9605034

Draper TG (2000) Addition on a quantum computer https://doi.org/10.48550/ARXIV.QUANT-PH/0008033

Durr C, Hoyer P (1999) A quantum algorithm for finding the minimum. arXiv:quant-ph/9607014

Grover LK (1996) A fast quantum mechanical algorithm for database search https://doi.org/10.48550/ARXIV.QUANT-PH/9605043. publisher: arXiv Version Number: 3

Hamzadayi A, Yildiz G (2017) Modeling and solving static m identical parallel machines scheduling problem with a common server and sequence dependent setup times. Comput Ind Eng 106:287–298. https://doi.org/10.1016/j.cie.2017.02.013

IBM (2024) IntegerComparator (latest version) | IBM Quantum Documentation. https://docs.quantum.ibm.com/api/qiskit/qiskit.circuit.library.IntegerComparator

Karger D, Stein C, Wein J (2010) Scheduling algorithms p 20

Katoch S, Chauhan SS, Kumar V (2021) A review on genetic algorithm: past, present, and future. Multimed Tools Appl 80(5):8091–8126. https://doi.org/10.1007/s11042-020-10139-6 (http://link.springer.com/10.1007/s11042-020-10139-6)

Korte B, Vygen J (2018) Combinatorial optimization: theory and algorithms, algorithms and combinatorics, vol 21. Springer, Berlin. https://doi.org/10.1007/978-3-662-56039-6

Kravchenko S, Werner F (1997) Parallel machine scheduling problems with a single server. Math Comput Model 26(12):1–11. https://doi.org/10.1016/S0895-7177(97)00236-7

Lahoz-Beltra R (2016) Quantum genetic algorithms for computer scientists. Computers 5(4):24. https://doi.org/10.3390/computers5040024 (https://www.mdpi.com/2073-431X/5/4/24)

Lawler EL, Lenstra JK, Rinnooy Kan AH, Shmoys DB (1993) Chapter 9 sequencing and scheduling: Algorithms and complexity. In: Handbooks in Operations Research and Management Science, vol 4, Elsevier, pp 445–522, https://doi.org/10.1016/S0927-0507(05)80189-6

Almufti MS, Ahmad Shaban A, Arif Ali Z, Ismael Ali RA, Dela Fuente J (2023) Overview of metaheuristic algorithms. Polaris Global J Schol Res Trends 2(2), 10–3. https://doi.org/10.58429/pgjsrt.v2n2a144

Malossini A, Blanzieri E, Calarco T (2008) Quantum genetic optimization. IEEE Trans Evol Comput 12(2):231–241. https://doi.org/10.1109/TEVC.2007.905006 (http://ieeexplore.ieee.org/document/4358783/)

Nielsen MA, Chuang IL (2010) Quantum computation and quantum information, 10th edn. Cambridge University Press, Cambridge

Phillipson F (2024) Quantum computing in logistics and supply chain management an overview. https://doi.org/10.48550/arXiv.2402.17520, arXiv:2402.17520 [quant-ph]

SaiToh A, Rahimi R, Nakahara M (2014) A quantum genetic algorithm with quantum crossover and mutation operations. Quantum Inf Process 13(3):737–755. https://doi.org/10.1007/s11128-013-0686-6 (http://link.springer.com/10.1007/s11128-013-0686-6)

Shor PW (1997) Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Rev 26(5):1484–1509

Thakoor N, Devarajan V, Gao J (2009) Computation complexity of branch-and-bound model selection. In: 2009 IEEE 12th international conference on computer vision, IEEE, Kyoto, pp 1895–1900, https://doi.org/10.1109/ICCV.2009.5459420

Theradapuzha Mathew N, Johansson B (2023) Production planning and scheduling challenges in the engineer-to-order manufacturing segment-a literature study. Int J Innov Manage Technol 14:80–87. https://doi.org/10.18178/ijimt.2023.14.3.942

Thula (2023) Efficient quantum comparator circuit. https://egrettathula.wordpress.com/2023/04/18/efficient-quantum-comparator-circuit/

Udrescu M, Prodan L, Vladutiu M (2006) Implementing quantum genetic algorithms: a solution based on Grover's algorithm. In: Proceedings of the 3rd conference on computing frontiers, ACM, Ischia Italy, pp 71–82, https://doi.org/10.1145/1128022.1128034, https://dl.acm.org/doi/10.1145/1128022.1128034