Production Engineering (2023) 17:105–115
https://doi.org/10.1007/s11740-022-01145-8

PRODUCTION MANAGEMENT

Solving flexible job shop scheduling problems in manufacturing with Quantum Annealing

Philipp Schworm[1] · Xiangqian Wu[1] · Moritz Glatt[1] · Jan C. Aurich[1]

Received: 4 April 2022 / Accepted: 24 June 2022 / Published online: 27 July 2022

Abstract
Quantum Annealing (QA) is a metaheuristic for solving optimization problems in a time-efficient manner. Therefore, quantum mechanical effects are used to compute and evaluate many possible solutions of an optimization problem simultaneously. Recent studies have shown the potential of QA for solving such complex assignment problems within milliseconds. This also applies for the field of job shop scheduling, where the existing approaches however focus on small problem sizes. To assess the full potential of QA in this area for industry-scale problem formulations, it is necessary to consider larger problem instances and to evaluate the potentials of computing these job shop scheduling problems while finding a near-optimal solution in a time-efficient manner. Consequently, this paper presents a QA-based job shop scheduling. In particular, flexible job shop scheduling problems in various sizes are computed with QA, demonstrating the efficiency of the approach regarding scalability, solutions quality, and computing time. For the evaluation of the proposed approach, the solutions are compared in a scientific benchmark with state-of-the-art algorithms for solving flexible job shop scheduling problems. The results indicate that QA has the potential for solving flexible job shop scheduling problems in a time efficient manner. Even large problem instances can be computed within seconds, which offers the possibility for application in industry.

Keywords Quantum Annealing · Scheduling · Job shop scheduling · Optimization

Abbreviations

JSS	Job shop scheduling
JSSP	Job shop scheduling problem
FJSSP	Flexible job shop scheduling problem
DJSSP	Dynamic job shop scheduling problem
PPC	Production planning and control
QA	Quantum Annealing
QPU	Quantum processing unit
BQM	Binary quadratic model
DQM	Discrete quadratic model
CQM	Constrained quadratic model
CHS	Classical hybrid solver
HBQM	Leap hybrid BQM solver
HDQM	Leap hybrid DQM solver
HCQM	Leap hybrid CQM solver

✉ Philipp Schworm
philipp.schworm@mv.uni-kl.de

[1] Institute for Manufacturing Technology and Production Systems, TU Kaiserslautern, P.O. Box 3049, 67653 Kaiserslautern, Germany

1 Introduction and state of the art

The concept of Industry 4.0 is closely linked to the objective of economical and flexible production of customized products in small batch sizes. To meet this objective, unexpected failures of machines in a production system or disruption errors in supply chains e.g., caused by a pandemic situation or sudden disturbances in supply chains have to be considered. An essential aspect that is affected by these influences is production planning and control (PPC) [1]. This includes, among other steps, the planning of material requirements, the scheduling of orders, and capacity planning. The goals are a reduction of work in progress, a minimization of processing times, a reduction in inventory costs, or the ability to react to changes in demand or supply [2]. Therefore, PPC has to be dynamic, adaptive, and integrative and has to consider material requirements planning, enterprise resource planning, just-in-time manufacturing, and collaborative planning, forecasting, and replenishment, among other activities [3].

In order to meet these requirements, PPC needs tools for decision-making and planning. Thus, process scheduling, also known as job shop scheduling (JSS), is a crucial task of the PPC. JSS aims at determining the chronological processing sequence of given orders. In this process, the goal is to allocate a set of tasks (jobs) to available functional units (machines) as efficiently as possible with regards to a certain objectives [4]. These objectives can be e.g., minimizing the makespan (i.e., the completion time of all the jobs), minimizing the tardiness of each job [5], minimizing the energy consumption [6], or maximizing machine utilization [7]. Objectives can be considered individually or on a multi-criteria basis. In consequence, optimization problems can be formulated, which range under the term of job shop scheduling problem (JSSP). The general assumption is that the operations of a job have to be processed in a given order and a machine cannot process two operations at the same time, which builds constraints of the JSSP. Constraints and objectives vary between JSSP types. One important expansion of the JSSP towards industrial applicability is the flexible job shop scheduling problem (FJSSP), in which operations can be processed by more than one machine [8]. Furthermore, in the dynamic job shop scheduling problem (DJSSP) e.g., availability states of machines are considered [9].

In order to solve these various optimization problems, computer-aided methods like optimization algorithms can be used. Using exact optimization methods for NP-hard problems such as JSSP is linked with an exponential growth in runtime with the problem size. Therefore, approximation methods are deployed to find solutions, since exact methods are mostly not able to compute these in a time-efficient manner [10]. Mokhtari and Hasani used a combination of genetic and simulated annealing algorithms in order to solve multi-objective FJSSP [11]. Besides, Zhang et al. used a digital twin to improve the solution of a DJSSP. The digital twin is used as a basis for forecasting machine failures, as well as for intelligent JSS by means of a genetic algorithm [12]. These methods show capabilities for finding solutions in a time efficient manner. However, the computational effort increases rapidly with the problem size even with approximation methods. Furthermore, current developments in the field of Quantum Annealing (QA) show huge potential for application.

QA is a metaheuristic, which is proposed showing advantages solving combinatorial optimization problems compared to algorithms on classical computers [13]. Therefore, quantum annealers based on the adiabatic theorem, are realized in order to make QA applicable. Basic units of a quantum annealer are quantum bits or qubits, which describe the lowest information unit in the quantum processing unit (QPU). Similar to bits of classical computers, qubits can attain the states of 0 or 1. However, qubits are quantum objects, which results in the possibility of assuming an infinite number of states between 0 and 1 at the same time. This phenomenon is known as *superposition* [14]. Before a QA process is initiated, the qubits are in superposition. The superposition ends when the QA process is finished and the qubits collapse to either 0 or 1. The probability in which state a qubit collapses can be influenced by biases applying external magnetic fields. In addition, through the quantum physical phenomenon of *entanglement*, qubits can be linked together so that they influence each other. Entanglement can be controlled analog to biases by couplers that apply external magnetic fields during the QA. Through couplers, the end states of entangled qubits are influenced [15]. During QA, an energy landscape for qubits is defined through couplers and biases in which the quantum annealer finds the minimum energy state. These states can be assigned to possible solutions of a minimization problem, where low energy states are linked with good solutions. In finding an optimal solution along the energy profile, states of higher energy usually have to be overcome to find a state of lower energy. However, during QA, *quantum tunneling* makes it possible to find the lowest energy level by passing through higher energy levels [16]. This procedure is shown in Fig. 1, which summarizes the QA process. Based on the described effects, QA is able to represent many solutions of a combinatorial optimization problem at the same time through superposition and finds a good solution through tunneling and entanglement within milliseconds, even for large problem sizes [17]. In difference to gate-based quantum computers the latest versions of quantum annealers have over 5000 Qubits and are already able to address realistic problem sizes. In addition, the useability of QA for industrial applications came along with the possibility of controlling QA via cloud services by providers such as D-Wave[1] [18]. For example, to show the potential of QA for industrial applicability an approach for factory layout planning using QA was proposed by Klar et. al [19].

The inputs for quantum annealers are specific formulations of an energy optimization problem in the form of Ising models. Many optimization problems can be formulated in this way using the formulation as a hamilton function [20]. A hamilton function (Eq. 1) maps certain states of an optimization problem to their specific energy levels. The hamilton function is described through the sum of the initial hamilton and the final hamilton, also called tunneling and problem hamilton.

$$H_{ising} = A(s)H_{inital} + B(s)H_{final} \tag{1}$$

$A(s)$ and $B(s)$ are energy scaling functions that increase or decrease monotonically with the progress of the QA. At the beginning of a QA process, all qubits are in superposition and the system is in its lowest energy state mainly described by the initial hamilton (i.e., $A(s) = 1, B(s) = 0$). During the

[1] Naming of specific company is done solely for the sake of completeness and does not necessarily imply an endorsement of the named companies nor that the products are necessarily the best for the purpose.

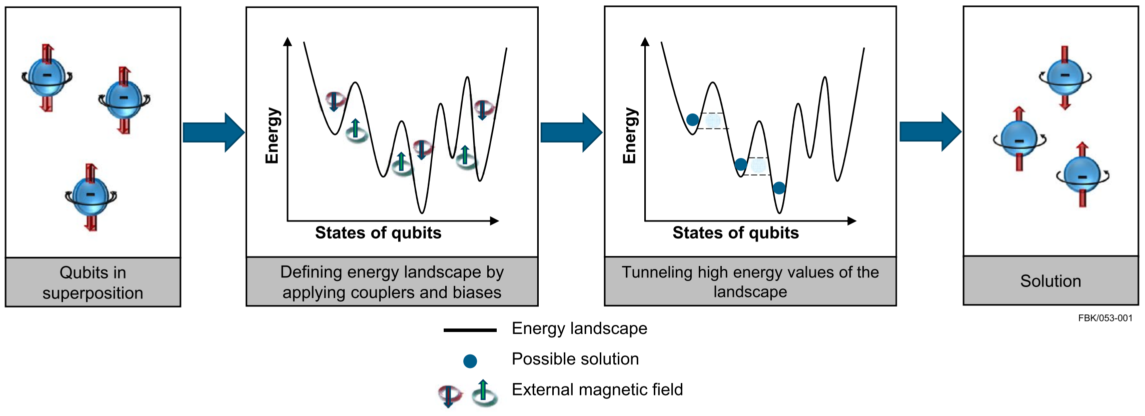

Fig. 1 Quantum Annealing process

QA the influence of the initial hamilton decreases and the final hamilton increases. The lowest energy state of the final hamilton describes the solution of the minimization problem, including qubit biases and couplings. At the end of QA process (i.e., $A(s) = 0, B(s) = 1$), the qubits remain in a state described by the final Hamilton, which can be applied to annealers formulated as [18]:

$$H_{final} = \sum_i Q_{ii}x_i + \sum_{i<j} Q_{ij}x_ix_j \tag{2}$$

with scalar weights Q_{ii}, Q_{ij} and binary variables x_i, x_j.

Recent studies regarding JSS have shown the potential of QA to solve such complex assignment problems within milliseconds using the hamilton formulation [21, 22]. QA offers the potential to compute the JSSP while finding a near-optimal solution in a time-efficient manner. Thus, Venturelli et al. proposed a valuable approach for solving small JSSP under finding optimal solutions [21]. The feasibility to solve JSSP with QA could successfully be shown in this approach. However, the full potential of QA should be explored by testing larger problem sizes. Besides, Kurowski et al. proposed how JSSP can be decomposed into a set of smaller optimization problems that requires less quantum hardware capacity [22], which makes the solution of larger problems possible. Aiming at industrial scale problems, Denkena et al. use a digital annealer, which simulates the principles of a quantum annealer, to solve larger instances of FJSSP [23]. Especially through the solutions of the considered problem sizes this approach shows perspectives for application in industry. Nevertheless, digital annealers only simulate quantum mechanical effects. Therefore, it can be assumed that a quantum annealer with an adequate number of qubits performs better in solving problem instances for industrial applicability regarding computation time.

Consequently, this paper presents a QA-based FJSSP for varying problem sizes. In a first step, the mathematical formulation for mapping FJSSP to a quantum annealer will be shown. Furthermore, the different solvers of QA are evaluated through a scientific benchmark to demonstrate the efficiency of the approach regarding scalability, solutions quality, and computing time.

2 Problem formulation and solver description

2.1 Framework

In this paper, the QA-based job shop scheduling approach is presented to solve FJSSP for different job sizes using D-Wave solvers. Where the QPU solvers only use the QPU of the quantum annealers, the hybrid solvers use both classical and quantum resources to solve problems. In this paper only hybrid solvers are applied, which are suitable for solving large problem instances. D-Wave offers access to leap hybrid solvers and classical hybrid solvers (CHS). The leap hybrid solvers support different kinds of quadratic models as input. Sets of binary variables defined in a hamilton formulation as binary quadratic model (BQM) are suitable for the leap hybrid BQM solver (HBQM). In contrast, discrete variables, which can assume e.g., integers, are combined in a discrete quadratic model (DQM) that the leap hybrid DQM solver (HDQM) supposes. Besides, constrained quadratic models (CQM) containing integer and binary variables are required for the leap hybrid CQM solver (HCQM). CHS support only BQM. Where the leap hybrid solvers use the QPU only once for computing a submitted problem, the CHS are programmed to decompose the problem into smaller instances and compute iteratively.

The proposed approach aims at finding feasible schedules for given sets of jobs and machines in a time-efficient manner so that multiple schedules can be computed, and the best solution can be evaluated (shown in Fig. 2). In addition, it will be examined which solver is best suited for the various

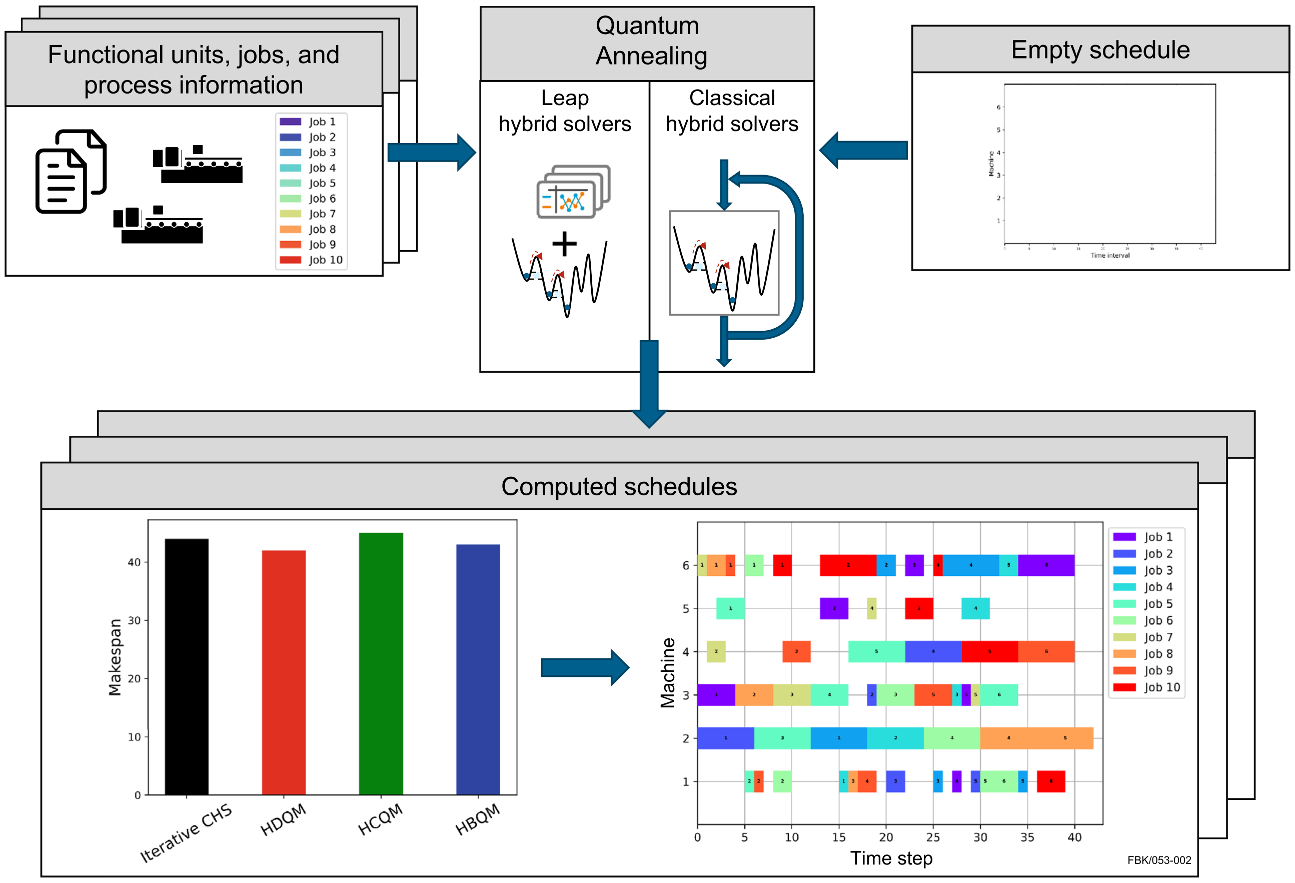

Fig. 2 Proposed framework

problem sizes by testing different solvers. Therefore, the leap solvers will be used as well as one CHS. Though the CHS support the decomposer to split large problems into multiple small sub-problems, an additional iterative approach is proposed to achieve faster computing time by using a CHS. As the starting point, it is essential for the QA approach to determine the input variables, constraint conditions and objectives, and summarize them in a mathematical formulation.

FJSSP aims to schedule A jobs $J = \{j_1, \cdots, j_A\}$ on B machines $M = \{m_1, \cdots, m_B\}$ with given optimization objectives. Each job consists of various operations that must be performed in a predefined sequence. O_i is denoted as the set of operations for any job $i \in J$. Furthermore, any operation $o_i \in O_i$ can be processed on at least one machine out of the given set M_{o_i}. For any $o_i \in O_i$ and $m \in M_{o_i}$, the processing time of the operation on machine is defined as $p_{o_i,m}$ and the starting time of the operation is denoted as t, which is in the given timeline $T = \{0, \dots, T_{max}\}$.

Additionally, the solution for the FJSSP should satisfy the following three constraints:

1. **Processing constraint:** During the processing of any job $i \in J$, each operation $o_i \in O_i$ must start only once on a single machine $m \in M_{o_i}$.
2. **Procedure constraint:** For any job $i \in J$, each operation $o_i \in O_i$ must be processed in the given order $L_{O_i} = \{0, \dots, l_{o_i}, \dots, l_{o_{i,last}}\}$, in which $o_{i,last}$ indicates the last operation of the job i.
3. **Overlapping constraint:** Each machine $m \in M$ cannot process more than one operation at the same time.

The objectives for FJSSP can be varied (e.g., energy consumption, job completion time and processing costs). However, the proposed approach focuses on the optimization of job completion time. Therefore, only one optimization objective is implemented, the makespan objective.

2.2 Mathematical formulation

In order to exploit and evaluate the full potential of the various solvers for computing FSJP, the constraints and objectives have to be formulated using the specific kind of supported variables.

2.2.1 BQM formulation

In the BQM formulation (Table 1) according to [23], a set of binary variables is used to denote all the possible starting times of each operation on the corresponding machine. The binary variable $k_{o_i,m,t}$ is equal to 1 if the referred operation $o_i \in O_i$ starts on the corresponding machine $m \in M_{o_i}$ to the allocated discrete time $t \in T$ (Eq. 3). The constraints and objectives are defined as binary polynomials in a hamilton formulation and summarized in a binary polynomial H (Eq. 11) which serves as a cost function for the optimization problem. The binary polynomials H_1, H_2, H_3, H_4 are added together with non-negative scalar weights $\alpha, \beta, \gamma, \delta$ which determine the impact of the respective polynomial. With H_1 a processing constraint is formulated (Eq. 4). Obviously, the constraint is satisfied for $H_1 = 0$. Here, combinations of binary variables which don't fulfill the constraint result in bigger values, e.g., if an operation has multiple starting times. The procedure constraint is fulfilled analog to the first constraint for $H_2 = 0$ (Eq. 5). Here the constraint is violated if the given order of a job is not considered. The third constraint leads to $H_3 = 0$ if no machine is occupied by two operations simultaneously. To accomplish the optimization objective of completing each task in the shortest time, a binary polynomial H_4 is defined to penalize the completion time late operations (Eq. 9). The makespan objective,H_4 penalizes the completion time of any operation that is later than minimum predecessor time of the operation P_{o_i}, which is the sum of the minimum processing times of the preceding operations of operation o_i (Eqs. 9, 10).

2.2.2 DQM formulation

In the DQM formulation (illustrated in Table 2), the start time of the operation can be determined by a discrete variable instead of a set of binary variables in the BQM formula. To assign the corresponding machine to the operation, v_{o_i} is an integer in the range from 0 to $N_{o_i} \cdot T_{max}$, where N_{o_i} is the number of selectable machines for the operation and T_{max} is the defined maximum completion time (Eq. 15). The starting time of the operation on the machine is determined using the following equation:

$$x_{o_i,m} = v_{o_i} - \left(n_{o_i,m} - 1\right)T_{max} \tag{12}$$

where $n_{o_i,m} \in [1, \dots, N_{o_i}]$ indicates the machine number of the assigned machine $m \in M_{o_i}$ in the available machines for operation o_i.

As in the BQM formulation, a binary variable is used in DQM to determine if the operation starts on the indicated machine to the assigned discrete time (Eq. 13), and a polynomial H summarizes the constraints and objectives correspondingly (Eq. 22). The polynomials H_2, H_3, H_4, H_5 with non-negative scalar weights $\alpha, \beta, \gamma, \delta$ are added corresponding to the constraints and objectives. However, compared to BQM, DQM does not require processing constraints. Since the starting time of each operation is

Table 1 Constraints and objectives in BQM formulation

Variables	$k_{o_i,m,t} = \begin{cases} 1 : \text{operation } o_i \text{ starts on machine } m \in M_{o_i} \text{ at time } t \\ 0 : \text{otherwise} \end{cases}$	(3)
Processing constraint	$H_1 = \sum_{o_i \in O_i} \left(1 - \sum_{t \in T} \sum_{m \in M_{o_i}} k_{o_i,m,t}\right)^2$	(4)
Procedure constraint	$H_2 = \sum_{i \in J} \sum_{\substack{o_i, o'_i \in O_i \\ l_{o_i} < l_{o'_i}}} \sum_{\substack{t' - t < p_{o_i,m} \\ (m,m') \in M_{o_i} \times M_{o'_i}}} k_{o_i,m,t} \cdot k_{o'_i,m',t'}$	(5)
Overlapping constraint	$H_3 = \sum_{m \in M_{o_i} \cap M_{o_j}} \sum_{\substack{(o_i,o_j) \in O_i \times O_j \\ (i,j,t,t') \in G \cup H}} k_{o_i,m,t} \cdot k_{o_j,m,t'}$	(6)
	where	
	$G = \left\{ (i,j,t,t') : i,j \in J, i \neq j, t, t' \in T, 0 \leq t - t' < p_{o_j,m} \right\}$	(7)
	$H = \left\{ (i,j,t,t') : i,j \in J, i \neq j, t, t' \in T, 0 \leq t' - t < p_{o_i,m} \right\}$	(8)
Makespan objective	$H_4 = \sum_{\substack{o_i \in O_i \\ m \in M_{o_i}}} k_{o_i,m,t} \cdot \left(t + p_{o_i,m} - P_{o_i}\right)$	(9)
	where	
	$P_{o_i} = \sum_{l_{o'_i} < l_{o_i}} \min_{m' \in M_{o'_i}} p_{o'_i,m'}$	(10)
Objective function	$H = \alpha H_1 + \beta H_2 + \gamma H_3 + \delta H_4$	(11)

Table 2 Constraints and objectives in DQM formulation

Binary variable	$c_{x_{o_i,m}} = \begin{cases} 1: \text{operation } o_i \text{ starts on machine } m \in M_{o_i} \text{ at time } x_{o_i,m} \\ 0: \text{otherwise} \end{cases}$	(13)
Discrete variables	$x_{o_i} \in \left[0, \cdots, N_{o_i} \cdot T_{max}\right]$	(14)
	$T_{sum} \in T$	(15)
Processing constraint	–	
Procedure constraint	$H_2 = \sum_{i \in J} \sum_{\substack{o_i, o'_i \in O_i \\ l_{o_i} < l_{o'_i}}} \sum_{\substack{t' - t < p_{o_i,m} \\ (m,m') \in M_{o_i} \times M_{o'_i}}} c_{o_i,m} \cdot c_{o'_i,m'}$	(16)
Overlapping constraint	$H_3 = \sum_{m \in M_{o_i} \cap M_{o_j}} \sum_{\substack{(o_i,o_j) \in O_i \times O_j \\ \left(i,j,x_{o_i,m},x_{o_j,m}\right) \in B \cup I}} c_{x_{o_i,m}} \cdot c_{x_{o_j,m}}$	(17)
	where	
	$B = \left\{(i,j,x_{o_i,m},x_{o_j,m}) : i,j \in J, i \neq j, x_{o_i,m}, x_{o_j,m} \in T, 0 \leq x_{o_i,m} - x_{o_j,m} < p_{o_j,m}\right\}$	(18)
	$I = \left\{(i,j,x_{o_i,m},x_{o_j,m}) : i,j \in J, i \neq j, x_{o_i,m}, x_{o_j,m} \in T, 0 \leq x_{o_j,m} - x_{o_i,m} < p_{o_i,m}\right\}$	(19)
Makespan constraint	$H_4 = \sum_{i \in J} \sum_{\substack{m \in M_{o_{i,last}} \\ x_{o_{i,last},m} + p_{o_{i,last},m} > T_{sum}}} T_{sum} \cdot c_{x_{o_{i,last},m}}$	(20)
Makespan objective	$H_5 = T_{sum}$	(21)
Objective function	$H = \alpha H_2 + \beta H_3 + \gamma H_4 + \delta H_5$	(22)

already defined by Eq. 14, no more than one machine can be assigned to an operation.

Moreover, to satisfy the makespan objective, a discrete variable T_{sum} is directly defined as makespan (Eq. 15) and a makespan constraint is added by the polynomial H_5. Here, if the completion time of the last operation of any task is greater than T_{sum}, the constraint is violated (Eq. 20). $H_5 = 0$ means that the makespan constraint is satisfied.

2.2.3 CQM formulation

Both BQM and DQM are unconstrained models that all constraints and objectives are contained in the polynomial. The optimal solution is found by controlling the scalar weights corresponding to the polynomial. In contrast, CQM is a constrained quadratic model, which sets the independent equations corresponding to the constraints. The objective is expressed in terms of minimizing a polynomial. Analog to DQM, CQM supports integer variables and binary variables. In the CQM expression (presented in Table 3), a binary variable is defined by an operation and the corresponding machine. If the operation is assigned to the corresponding machine, the variable equals 1 (Eq. 23). Another binary variable is defined by two operations and a machine, in order to determine whether two operations are assigned to the same machine (Eq. 24). In addition, the starting time of the operation is defined by a positive integer variable (Eq. 25). Similar to DQM, the makespan is determined by an integer variable (Eq. 26). In CQM, the processing constraint is formulated in Eq. 27. The sum of the binary variables $y_{o_i,m}$ of all the different machines for any one operation equals to 1.

To fulfill the processing constraint, the difference between the start time of operation and the next operation in the same job is greater than the duration time required to the operation (Eq. 28). Equation 29 forms the overlapping constraint. If two operations are assigned on the same machine, the difference between their starting times can not be less than the processing time of the previous executed operation. Moreover, the makespan objective minimizes the makespan that is defined by an integer variable (Eq. 31). Equation 30 for the makespan constraint is used to control the completion time of last operation for any job not to exceed the makespan.

2.2.4 Variable pruning

In addition to a minimum predecessor time P_{o_i}, non-final operations in any job have a minimum successor time S_{o_i}, which is the sum of the minimum processing durations of all subsequent operations including the processing duration of current operation o_i (Eq. 32).

$$S_{o_i} = \sum_{l_{o'_i} \geq l_{o_i}} \min_{m' \in M_{o'_i}} p_{o'_i,m'} \tag{32}$$

Table 3 Constraints and objectives in CQM formulation

Binary variables	$y_{o_i,m} = \begin{cases} 1 : \text{operation } o_i \text{ starts on machine } m \in M_{o_i} \\ 0 : \text{otherwise} \end{cases}$	(23)
	$z_{o_i,o_j,m} = \begin{cases} 1 : \text{operations } o_i, o_j \text{ start on the machine } m \in M_{o_i} \cap M_{o_j} \\ 0 : \text{otherwise} \end{cases}$	(24)
Integer variables	$x_{o_i} \in T$	(25)
	$T_{sum} \in T$	(26)
Processing constraint	$\sum_{i \in J} \sum_{m \in M_{o_i}} y_{o_i,m} = 1$	(27)
Procedure constraint	$y_{o_i,m}\left(x_{o_i} - x_{o_i'}\right) \geq y_{o_i,m} \cdot p_{o_i,m}$ where $x_{o_i}, x_{o_i'} \in T, m \in M_{o_i}, l_{o_i} < l_{o_i'}, i \in J, o_i, o_{i'} \in O_i$	(28)
Overlapping constraint	$z_{o_i,o_j,m}\left(x_{o_j} - x_{o_i} - p_{o_i,m}\right)\left(x_{o_i} - x_{o_j} - p_{o_j,m}\right) \leq 0$ where $x_{o_i}, x_{o_j} \in T, m \in M_{o_i} \cap M_{o_j}, i, j, \in J, o_i, o_j \in O_i \times O_j$	(29)
Makespan constraint	$T_{sum} \geq x_{o_{i,last}} + p_{o_{i,last},m}$ where $x_{o_{i,last}} \in T, m \in M_{o_{i,last}}, i \in J, o_{i,last} \in O_i$	(30)
Makespan objective	$\min\left(T_{sum}\right)$	(31)

The starting time of the operation o_i should be in the range of $[P_{o_i}, T_{max} - S_{o_i}]$. Therefore, variables that are not in this time range in BQM can be pruned directly to reduce the computation time and decrease the size of the computational problem. Analog in the DQM formulation, the range of the discrete variable can be pruned leading to less interactions between variables and smaller problem sizes. Furthermore, in the CQM, the set of binary variables can be reduced by non-feasible combinations of machines as well as non-valid starting times. In addition, for the parameter T_{max} a as small as possible value should be chosen in order to minimize the problem size for all solvers. Therefore, an estimation has to be done which depends on the number of operations in the given jobs, available machines, and the corresponding processing times. For each job, the processing times of the operations can be summarized. The maximum processing time through all jobs leads to a lower bound of T_{max}. Additionally, the sum of all jobs processing times leads to an upper bound for T_{max}. In order to use suitable values for the different problem sizes starting from the lower bound, the value will be increased with the number of operations as well as processing durations and decreased with the number of machines.

2.2.5 Iterative approach for large problems

In order to reduce the number of variables, an iterative approach is proposed by decomposing large problems into multiple small sub-problems in addition to the decomposing mechanic of the CHS mentioned above. The approach is illustrated in Fig. 3. As the starting point, the given problems are split into different samples to determine the number of jobs and corresponding operations scheduled in each iteration. The chosen sample sizes depend on the number of jobs. To fulfill the makespan objective, the jobs are selected in priority order according to the minimum processing times of all the operations. Moreover, the initial range of times that each job can be processed on the corresponding machine $T_{i,m}^{(0)}$ and scalar weight of the objective function are required to be defined. Therefore, an estimation according to chapter 2.2.4 has to be done. To reduce the number of variables, the initial maximum completion time is set, which is then gradually increased with each iteration and the processing times of the operations that have

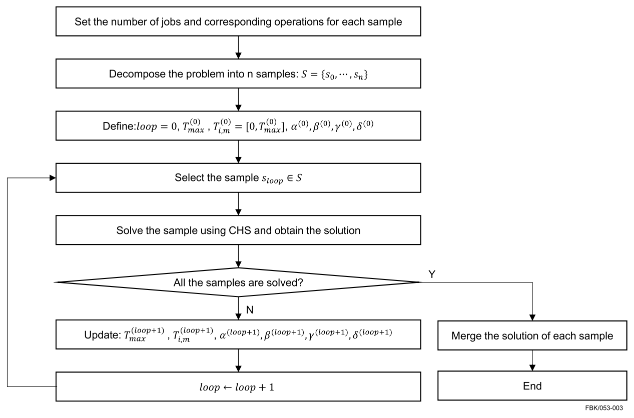
Set the number of jobs and corresponding operations for each sample
Decompose the problem into n samples: $S = \{s_0, \cdots, s_n\}$
Define: $loop = 0, T_{max}^{(0)}, T_{i,m}^{(0)} = [0, T_{max}^{(0)}], \alpha^{(0)}, \beta^{(0)}, \gamma^{(0)}, \delta^{(0)}$
Select the sample $s_{loop} \in S$
Solve the sample using CHS and obtain the solution
All the samples are solved?
Y
N
Update: $T_{max}^{(loop+1)}, T_{i,m}^{(loop+1)}, \alpha^{(loop+1)}, \beta^{(loop+1)}, \gamma^{(loop+1)}, \delta^{(loop+1)}$
Merge the solution of each sample
$loop \leftarrow loop + 1$
End
FBK/053-003

Fig. 3 Workflow of the iterative approach

already been scheduled are removed. Consequently, T_{max} is updated if the estimated completion time of the remaining jobs in the loop is beyond T_{max}. Afterwards, each sample is solved using the CHS. The iteration is completed until all the samples are completed. The solution to the problem is the combination of the solutions of all the samples in the loop.

3 Benchmark

For evaluation of the different approaches, various FJSSP were computed, and the solvers were examined regarding performance and solution quality. Therefore, FJSSP were computed with the different hybrid QA solvers under consideration of various problem instances. Hence, statements could be made which QA-based solver is best suitable to which problem sizes. For that purpose, the input parameters of the solver were adjusted until feasible solutions could be found. These results are shown in Table 4 in which "–" expresses that the solution is not found.

The results of computing various FJSSP show that very small problem instances can be solved by every solver under finding good solutions. However, the iterative CHS needs significantly less computation time than the other hybrid solvers for these instances. Since the leap hybrid solvers like HDQM and HCQM can't go below a minimum computing time of five seconds due to input parameter restrictions, the iterative CHS shows advantages over the leap hybrid solvers when computing small problem instances regarding computing time. In addition, the solution quality of all solvers is comparable which leads to advantages using iterative CHS for small problem instances. With increasing problem sizes, the computing time for the leap hybrid solvers initially remains the same and starts to increase noticeably only from the 20 × 10 × 10 instance. In comparison, the computation time of the iterative CHS does not increase with the size of the problem due to the iteration approach. The computing time of the iterative CHS is affected by the number of iterations, which also has an influence on the quality of the solution. Therefore, the computing time for the large problem instances (30 × 20 × 10 and 30 × 20 × 15) is essentially the same as the small problem instances (3 × 3 × 3 and 6 × 6 × 6). The solution quality is comparable yet for the HDQM, the HBQM, and the iterative CHS. However, the solution quality for the HCQM worsens in comparison with the other hybrid solvers. For some instances the HCQM does not even find a solution because the permissible variable number is exceeded. It can be concluded that for medium-sized instances, the HDQM as well as HBQM show the highest suitability for finding good solutions, while the iterative CHS can be used for evaluating many solutions due to the significantly lower computing time. Moreover, the computing time for solving medium sized instances increases faster with the HBQM than with the HDQM. Consequently,

Table 4 Results of various FJSSP

	HCQM	HDQM	Iterative CHS		HBQM
Problem size[1]	Solution[2]	Solution[2]	Solution[2]	No. of Iterations	Solution[2]
3 × 3 × 3	8/5	8/5.00	8/0.23	1	8/3.02
6 × 6 × 6	18/5	17/5.00	16/0.27	4	16/3.42
8 × 8 × 8	–	22/5.00	19/0.36	5	19/3.65
10 × 15 x 10	–	38/6.01	42/0.50	7	38/12.02
20 × 10 x 10	–	62/18.63	64/0.61	15	59/52.68
20 × 15 x 10	–	61/21.12	75/0.20	4	56/51.20
20 × 15 x 15	–	52/10.05	68/0.14	3	57/48.41
30 × 20 x 10	–	113/153.67	146/0.26	6	158/282.30
30 × 20 x 15	–	85/81.01	107/0.27	6	–

[1]Jobs × operations × machines

[2]Makespan/Computing time (s)

for the 20 × 15 × 15 instance, the computing time is almost five times higher. Regarding large problem instances (30 × 20 × 10 and 30 × 20 × 15), the HDQM requires higher computing times as well. While the solution times of the iterative CHS are still in the range of milliseconds or seconds, the solution time for the leap hybrid solvers is in the scope of minutes. Regarding the solution quality, the HDQM produces the best solutions, where the HBQM only reaches poor makespan values. The solution quality of the iterative CHS is lower than the HDQM solution, but under consideration of the low computing time, many solutions adapting the solver parameters can be achieved and evaluated in a time efficient manner. The computing times are shown in Fig. 4.

Since the HDQM and iterative CHS achieved the best results regarding solution quality and computing times, they will be chosen for an additional scientific benchmark according to [24]. The benchmark includes ten FJSSP (MK01-MK10) with various amount of machines, operations, and processing times that has already been performed by several authors, e.g., by [23, 25–28]. The objective is the minimization of the makespan. For comparison, the results of the benchmark are illustrated in Table 5. Furthermore, solutions and computing times of different approaches are shown in Table 6. So, the comparison of the QA approaches with state-of-the-art algorithms is guaranteed. However, it has to be mentioned that the best-known solutions are computed with approximative methods because of the NP-hardness of the problem. Therefore, it might be possible that the best-known solution is not the optimal solution.

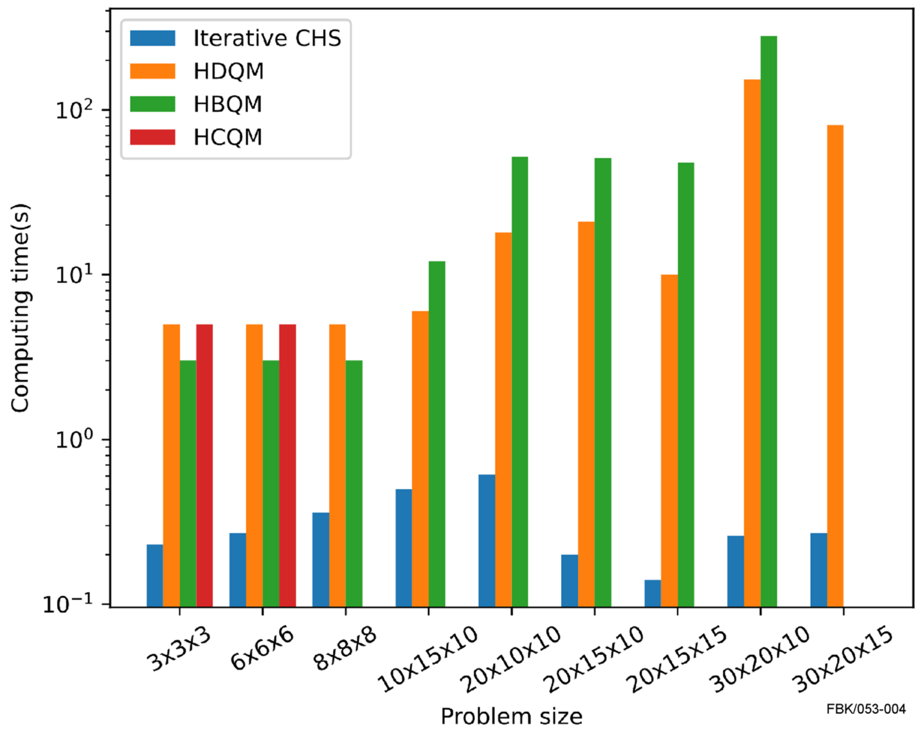

Fig. 4 Computing times for various FJSSP

According to Table 6, the approach proposed by [28] demonstrates a significantly higher computing time range from 3.02 to 122.52 min relative to other approaches. Though the approach using memetic algorithms proposed by [26] shows all the best-known solutions for 10 MK problems, the highest computing time for MK 06 reaches 80 s. Therefore, it is obvious that the proposed approach using iterative CHS in this paper has a great advantage in computing time compared to other proposed approaches. Even for the largest problems, the computing time is within 1 s. However, in the iterative approach, the jobs are prioritized according to the length of the required processing time, which can affect the solution quality. However, the results of the benchmark exhibit a good performance of the HDQM. It finds solutions for every MK-problem. For MK03 and MK08 it even achieves the best-known benchmark solution. Furthermore, no solution deviates from the best-known solution over 19%. The computing time increases with the problem instances, but is still much lower than the times e.g., [28] or [25] achieved.

In conclusion, QA bears potentials for solving FJSSP. In particular, computing many solutions in a short time offers the advantage of computing many solutions to a given problem and evaluating the solutions with respect to different objectives or constraints. Consequently, the QA-based job shop scheduling approach using hybrid solvers can solve FJSSP relative efficiently.

Table 5 Results of scientific benchmark

	DQM	Iterative CHS		Benchmark solution
Problem size	Solution[1]	Solution[1]	No. of Itera-tions	Solution[2]
MK01	42/6.08	42/0.43	6	37/40.6
MK02	31/9.01	28/0.34	5	26/26.6
MK03	204/221.36	213/0.35	5	204/204
MK04	66/12.14	69/0.26	5	60/63
MK05	176/60.03	180/0.40	6	172/174,2
MK06	67/90.09	71/0.43	4	57/63.8
MK07	153/112.56	152/0.38	5	139/144.4
MK08	523/220.10	549/0.64	14	523/523
MK09	317/297.23	333/0.58	14	307/310.2
MK10	225/273.12	241/0.57	13	197/221.8

[1]Makespan/Computing time (s)

[2]Best known benchmark solution/average value of solutions presented in Table 6

4 Conclusion and Outlook

Modern manufacturing in the context of Industry 4.0 increases the frequency and complexity of scheduling. In order to meet these requirements efficient algorithms for scheduling are needed. Consequently, this paper presents a QA-based approach for solving FJSSP. After presenting the framework of the approach, the mathematical

Table 6 Comparison of solutions from other proposed approaches

Problem size	Bagheri et al. [25] Solution[1]	Yuan and Xu [26] Solution[1]	Gao et al. [27] Solution[1]	Denkena et al. [23] Solution[1]	Xing et al. [28] Solution[2]
MK01	40/97.21	40/20.16	40/3.36	41/7.8	42/4.78
MK02	26/103.46	26/28.21	26/3.72	27/10.34	28/3.02
MK03	204/247.37	204/53.76	204/1.56	204/10.90	204/26.14
MK04	60/152.07	60/30.52	60/66.58	67/8.15	68/17.74
MK05	173/171.95	172/36.36	173/78.45	176/8.09	177/8.26
MK06	63/245.62	59/80.61	60/173.98	62/10.73	75/18.79
MK07	140/161.92	139/37.42	139/66.19	144/8.10	150/5.68
MK08	523/392.25	523/77.71	523/2.15	523/0,66	523/67.67
MK09	312/389.71	307/75.23	307/304.43	314/18.96	311/77.76
MK10	214/384.54	202/90.75	202/418.19	214/27.06	277/122.52

[1]Makespan/Computing time (s)

[2]Makespan/computing time (min)

formulations for the different kind of solvers are shown. Afterwards FJSSP with various sizes were solved using the HBQM, HDQM, and HCQM solver as well as iterative CHS. In this process, FJSSP are computed to analyze the suitability of the various problem sizes to the different solvers. Based on these results, a scientific benchmark with other state-of-the-art algorithms was performed with a makespan objective. The scientific benchmark has shown the huge potential of QA for solving FJSSP by solving problems within seconds or milliseconds under finding good solutions. Consequently, the presented approach demonstrated its ability to find high-quality solutions in a short time and can be used to generate different schedule variants that can be evaluated against each other. Nevertheless, efficient algorithms for solving FJSSP are still quite far away from industrial application. In industry, many dynamic factors have to be considered such as unexpected failures of machines in a production system or disruption errors in supply chains. Furthermore, jobs are gradually received and order inquiries fluctuate. This leads to a demand for algorithms with consideration of these dynamic conditions.

In future research, the complexity of the allocation problem will be increased, by incorporating additional boundary conditions such as unavailabilities of machines. Moreover, objectives such as energy costs or machine utilization will be incorporated to realize a multi-objective optimization problem. In addition, it will be investigated how the mathematical formulations must be adjusted for different problem types (e.g., dynamic, flexible, multi-objective). Also, the solution quality of the HDQM and the iterative CHS have to be improved. For this purpose, the approach by using HDQM will be enlarged through an iterative solving technique. In addition, it would be desirable to develop a method to estimate the gap between optimal and approximative solution. Therefore, further research based on the results will be developed incorporating techniques to choose suitable sample sets for the iterative approach. Since the iterative solution method of the BQM formulation has already been shown to improve computation time, the same approach can be expected for the iterative formulation of DQM problems. In addition, the iterative CHS will be enlarged through bottleneck factors according to [23] for improving the job selection method for each iteration. Besides, a software demonstrator will be developed to enable intuitive interaction for planners without extensive experience with the QA interface structure.

Acknowledgements This research was funded by the Ministerium für Wirtschaft, Verkehr, Landwirtschaft und Weinbau Rheinland-Pfalz—4161-0023#2021/0002-0801 8401.0012.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Bueno A, Godinho Filho M, Frank AG (2020) Smart production planning and control in the Industry 4.0 context: A systematic literature review. Comput Ind Eng 149:106774. https://doi.org/10.1016/j.cie.2020.106774
2. Stevenson M, Hendry LC, Kingsman† BG (2005) A review of production planning and control: the applicability of key concepts

to the make-to-order industry. Int J Prod Res 43:869–898. https://doi.org/10.1080/0020754042000298520
3. Manufacturing planning and control for supply chain management (2011) APICS/CPIM, certification. McGraw-Hill, New York
4. Pinedo ML (2016) Scheduling. Springer, Cham. https://doi.org/10.1007/978-3-319-26580-3
5. Gao K, Cao Z, Zhang Le, Chen Z, Han Y, Pan Q (2019) A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems. IEEE/CAA J Autom Sinica 6:904–916. https://doi.org/10.1109/JAS.2019.1911540
6. Roth S, Kalchschmid V, Reinhart G (2021) Development and evaluation of risk treatment paths within energy-oriented production planning and control. Prod Eng Res Devel 15:413–430. https://doi.org/10.1007/s11740-021-01043-5
7. Huang X, Guan Z, Yang L (2018) An effective hybrid algorithm for multi-objective flexible job-shop scheduling problem. Adv Mech Eng 10:1–14. https://doi.org/10.1177/1687814018801442
8. Li X, Gao L (2016) An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. Int J Prod Econ 174:93–110. https://doi.org/10.1016/j.ijpe.2016.01.016
9. Shahrabi J, Adibi MA, Mahootchi M (2017) A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. Comput Ind Eng 110:75–82. https://doi.org/10.1016/j.cie.2017.05.026
10. Zhang J, Ding G, Zou Y, Qin S, Fu J (2019) Review of job shop scheduling research and its new perspectives under Industry 4.0. J Intell Manuf 30:1809–1830. https://doi.org/10.1007/s10845-017-1350-2
11. Mokhtari H, Hasani A (2017) An energy-efficient multi-objective optimization for flexible job-shop scheduling problem. Comput Chem Eng 104:339–352. https://doi.org/10.1016/j.compchemeng.2017.05.004
12. Zhang M, Tao F, Nee A (2021) Digital Twin Enhanced Dynamic Job-Shop Scheduling. J Manuf Syst 58:146–156. https://doi.org/10.1016/j.jmsy.2020.04.008
13. Chancellor N (2017) Modernizing quantum annealing using local searches. New J Phys 19:23024. https://doi.org/10.1088/1367-2630/aa59c4
14. McGeoch CC (2014) Adiabatic quantum computation and quantum annealing: theory and practice. Synth Lect Quant Comput 5:1–93. https://doi.org/10.2200/S00585ED1V01Y201407QMC008
15. Lanting T, Przybysz AJ, Smirnov AY, Spedalieri FM, Amin MH, Berkley AJ, Harris R, Altomare F, Boixo S, Bunyk P, Dickson N, Enderud C, Hilton JP, Hoskinson E, Johnson MW, Ladizinsky E, Ladizinsky N, Neufeld R, Oh T, Perminov I, Rich C, Thom MC, Tolkacheva E, Uchaikin S, Wilson AB, Rose G (2014) Entanglement in a quantum annealing processor. Phys Rev X 4:21041. https://doi.org/10.1103/PhysRevX.4.021041
16. Cohen E, Tamir B (2014) D-Wave and predecessors: from simulated to quantum annealing. Int J Quantum Inform 12:1430002. https://doi.org/10.1142/S0219749914300022
17. Hauke P, Katzgraber HG, Lechner W, Nishimori H, Oliver WD (2020) Perspectives of quantum annealing: methods and implementations. Rep Prog Phys 83:54401
18. Johnson MW, Amin MHS, Gildert S, Lanting T, Hamze F, Dickson N, Harris R, Berkley AJ, Johansson J, Bunyk P, Chapple EM, Enderud C, Hilton JP, Karimi K, Ladizinsky E, Ladizinsky N, Oh T, Perminov I, Rich C, Thom MC, Tolkacheva E, Truncik CJS, Uchaikin S, Wang J, Wilson B, Rose G (2011) Quantum annealing with manufactured spins. Nature 473:194–198. https://doi.org/10.1038/nature10012
19. Klar M, Schworm P, Wu X, Glatt M, Aurich JC (2022) Quantum annealing based factory layout planning. Manuf Lett 32:59–62. https://doi.org/10.1016/j.mfglet.2022.03.003
20. Lucas A (2014) Ising formulations of many NP problems. Front Physics 2:23024. https://doi.org/10.3389/fphy.2014.00005
21. Venturelli D, Marchand DJJ, Rojo G (2015) Quantum Annealing Implementation of job-shop scheduling
22. Kurowski K, Węglarz J, Subocz M, Różycki R, Waligóra G (2020) Hybrid quantum annealing heuristic method for solving job shop scheduling problem. In: Krzhizhanovskaya VV, Závodszky G, Lees MH, Dongarra JJ, Sloot PMA, Brissos S, Teixeira J (eds) Computational Science—ICCS 2020. Springer, Cham, pp 502–515. https://doi.org/10.1007/978-3-030-50433-5_39
23. Denkena B, Schinkel F, Pirnay J, Wilmsmeier S (2021) Quantum algorithms for process parallel flexible job shop scheduling. CIRP J Manuf Sci Technol 33:100–114. https://doi.org/10.1016/j.cirpj.2021.03.006
24. Brandimarte P (1993) Routing and scheduling in a flexible job shop by tabu search. Ann Oper Res 41:157–183. https://doi.org/10.1007/BF02023073
25. Bagheri A, Zandieh M, Mahdavi I, Yazdani M (2010) An artificial immune algorithm for the flexible job-shop scheduling problem. Fut Gen Comput Syst 26:533–541. https://doi.org/10.1016/j.future.2009.10.004
26. Yuan Y, Xu H (2015) Multiobjective flexible job shop scheduling using memetic algorithms. IEEE Trans Automat Sci Eng 12:336–353. https://doi.org/10.1109/TASE.2013.2274517
27. Gao KZ, Suganthan PN, Chua TJ, Chong CS, Cai TX, Pan QK (2015) A two-stage artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job insertion. Expert Syst Appl 42:7652–7663. https://doi.org/10.1016/j.eswa.2015.06.004
28. Xing L-N, Chen Y-W, Yang K-W (2009) An efficient search method for multi-objective flexible job shop scheduling problems. J Intell Manuf 20:283–293. https://doi.org/10.1007/s10845-008-0216-z