Solving Job-Shop Scheduling Problems with QUBO-Based Specialized Hardware

Jiachen Zhang, Giovanni Lo Bianco, J. Christopher Beck

Department of Mechanical and Industrial Engineering, University of Toronto
{jasonzjc, giolb, jcb}@mie.utoronto.ca

Abstract

The emergence of specialized hardware, such as quantum computers and Digital/CMOS annealers, and the slowing of performance growth of general-purpose hardware raises an important question for our community: how can the high-performance, specialized solvers be used for planning and scheduling problems? In this work, we focus on the job-shop scheduling problem (JSP) and Quadratic Unconstrained Binary Optimization (QUBO) models, the mathematical formulation shared by a number of novel hardware platforms. We study two direct QUBO models of JSP and propose a novel large neighborhood search (LNS) approach, that hybridizes a QUBO model with constraint programming (CP). Empirical results show that our LNS approach significantly outperforms classical CP-based LNS methods and a mixed integer programming model, while being competitive with CP for large problem instances. This work is the first approach that we are aware of that can solve non-trivial JSPs using QUBO hardware, albeit as part of a hybrid algorithm.

Introduction

Many of the recent advances in hardware design have been aimed at specific computational tasks, including solving combinatorial optimization problems. These emerging technologies, which include adiabatic and gate-based quantum computers (Mohseni et al. 2017) and CMOS annealers (Aramon et al. 2019), represent a variety of designs and underlying models of computation. Nonetheless, many of the designs for combinatorial optimization target Ising models: problems formulated as an Ising model or equivalently as a Quadratic Unconstrained Binary Optimization (QUBO) model (Kochenberger et al. 2014).

For planning and scheduling research, it is still an open question whether the novel hardware platforms can be used to solve non-trivial problems. In this paper, we answer this question in the affirmative for the case of the job shop scheduling problem (JSP). While there has been work on solving JSP on QUBO hardware (Kurowski et al. 2020; Venturelli, Marchand, and Rojo 2016; Shimada, Shibuya, and Shibasaki 2021), these approaches have tended to only represent very small problems that are trivially solved to optimality with standard techniques. In this study we take two approaches: direct models that attempt to represent and solve the whole JSP on specialized hardware; and hybrid models which decompose the problem and exploit both traditional approaches and specialized hardware. We make three primary contributions:

1. We study two direct QUBO models of JSP, compare their scalability, and identify their limits on current hardware.
2. We propose a novel large neighborhood search approach (LNS) that uses operation ranks obtained from solving rank-based QUBO models on single machines to form a neighborhood within which a multi-machine model is used to search for better solutions. The proposed approach is compatible with both traditional optimization software and specialized hardware.
3. We obtain strong results on large JSPs using constraint programming (CP) to solve the multi-machine problem and novel hardware for one-machine problems. The results are competitive with CP alone and better than all the existing hardware-based methods and two classical CP-based LNS variants. To the best of our knowledge, this hybrid approach is the first one that is able to solve non-trivial JSPs on specialized hardware.

Background

Job-Shop Scheduling Problem

The NP-hard Job-shop Scheduling Problem (JSP) is considered to be one of the most computationally stubborn combinatorial optimization problems (Lawler et al. 1993). Most optimization approaches have been applied to JSP, including constraint programming (Baptiste, Le Pape, and Nuijten 2001), mixed integer programming (Manne 1960), heuristics (Adams, Balas, and Zawack 1988), meta-heuristics (Nowicki and Smutnicki 2005), and large neighborhood search (Carchrae and Beck 2009)

A job-shop scheduling problem consists of a set of operations $\mathcal{A} = \{o_1, \ldots, o_{|\mathcal{A}|}\}$ to be scheduled on M machines. Each operation o_i has a positive duration d_i. The operations are partitioned into N jobs $A_1, A_2, \ldots, A_N$ with $A_j \subseteq \mathcal{A}$ and $|A_j| = M$. Every job has to be processed on each machine exactly once, hence a job consists of M operations and the number of jobs is $N = \frac{|\mathcal{A}|}{M}$.

Operations in the same job must be processed in a defined order. Specifically, for every A_j we define $\sigma^j =$

$\langle \sigma_1^j, \sigma_2^j, \ldots, \sigma_M^j \rangle$ with $\sigma_h^j \in \{1, ..., M\}$ being the machine that must process the h-th operation of the job. For convenience, we denote the operation of job j on machine m by $o_{j,m}$, with its duration $d_{j,m}$, and we use d_{max} to represent the maximum duration of operations in $\mathcal{A}$.

No operations can overlap on the same machine (unary capacity machines). We also assume non-preemption, i.e., an operation that has started processing cannot be interrupted. The classical objective of JSP is to minimize the makespan, the latest completion time of the operations. We use JSP($|\mathcal{A}|$,M,d_{max}) to represent a JSP specification.

Quadratic Unconstrained Binary Optimization

A Quadratic Unconstrained Binary Optimization (QUBO) model is the following problem:

$$\min y = \frac{1}{2} \sum_i \sum_{j \neq i} W_{i,j} x_i x_j + \sum_i b_i x_i + c, \qquad (1)$$

where $x \in \{0,1\}^n$ is a vector of binary decision variables, $W \in \mathbf{M}_{n,n}(\mathbb{R})$ is a symmetric weight matrix, $b \in \mathbb{R}^n$ is a bias vector, and $c \in \mathbb{R}$ is a constant. QUBO has been used to represent practical problems in combinatorial scientific computing (Shaydulin et al. 2019) and machine learning (Cohen, Senderovich, and Beck 2020). Most importantly here, a number of recent hardware designs use quantum or classical computing to directly solve QUBO problems in hardware (Venturelli, Marchand, and Rojo 2016; Coffrin, Nagarajan, and Bent 2019).

Direct QUBO Models for JSP

Our starting point is to evaluate the utility of using simple, "direct" formulations of JSP in the QUBO formalism. While, as we show, these models perform poorly, their weaknesses provide the motivation and justification to investigate more sophisticated approaches.

We present two direct QUBO formulations for JSP based on two common mixed integer programming (MIP) models of JSP: the disjunctive and the time-indexed models.

Disjunctive QUBO Model

The disjunctive QUBO (DQ) model is based on the disjunctive MIP model (Manne 1960). Variable $\hat{x}_{j,m}$ represents the integer start time of the operation of job j on machine m, while $\hat{C}_{max}$ is the integer makespan variable. Variable $z_{j,k,m}$ is a binary variable that is 1 if the operation of job j on machine m precedes the operation of job k on machine m and is 0 otherwise. $\hat{s}_{j,h}^{(1)}, \hat{s}_j^{(2)}, \hat{s}_{j,k,m}^{(3,1)}, \hat{s}_{j,k,m}^{(3,2)}$ are integer slack variables used in different constraints.

Every non-negative integer variable is represented as its binary expansion. For example, the binary representation of C_{max} is given by a sequence of binary bits $i^{(k)} \in \{0,1\}, \forall k = 1, ..., L(C_{max})$ such that

$$C_{max} = \sum_{k=1}^{L(C_{max})} 2^{k-1} i^{(k)}. \qquad (2)$$

We require $L(C_{max}) = \lfloor \log_2 C_{max} \rfloor + 1$ binary bits to represent C_{max}. For convenience, we use $\hat{C}_{max}$ to represent the binary expansion of C_{max}. And we use the same notation $\hat{x}$ for every integer variable x to refer to its binary expansion. The DQ is as follows.

$$\min_{\hat{x}, z, \hat{s}} \quad \hat{C}_{max} \qquad (3a)$$

$$+ p_1 \cdot \sum_{j=1}^{N} \sum_{h=2}^{M} \left(\hat{x}_{j,\sigma_{h-1}^j} + d_{j,\sigma_{h-1}^j} + \hat{s}_{j,h}^{(1)} - \hat{x}_{j,\sigma_h^j} \right)^2 \qquad (3b)$$

$$+ p_2 \cdot \sum_{j=1}^{N} \left(\hat{x}_{j,\sigma_M^j} + d_{j,M} + \hat{s}_j^{(2)} - \hat{C}_{max} \right)^2 \qquad (3c)$$

$$+ p_3 \cdot \sum_{m=1}^{M} \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} \left(\hat{x}_{k,m} - \hat{x}_{j,m} - d_{j,m} + V - V \cdot z_{j,k,m} - \hat{s}_{j,k,m}^{(3,1)} \right)^2 + \left(\hat{x}_{j,m} - \hat{x}_{k,m} - d_{k,m} + V \cdot z_{j,k,m} - \hat{s}_{j,k,m}^{(3,2)} \right)^2 \qquad (3d)$$

$$z_{j,k,m} \in \{0,1\}, \ \forall j, k = 1, ..., N, j < k, \ \forall m = 1, ..., M, \qquad (3e)$$

$$\hat{C}_{max}, \hat{x}_{j,m}, \hat{s}_{j,h}^{(1)}, \hat{s}_j^{(2)}, \hat{s}_{j,k,m}^{(3,1)}, \hat{s}_{j,k,m}^{(3,2)} \geq 0, \quad \forall j, k = 1, ..., N, j < k, \forall m = 1, ..., M. \qquad (3f)$$

Term (3a) is the original objective function of the JSP. Term (3b) is the quadratic representation of precedence constraints, added to the objective function as a penalty term. A precedence constraint guarantees the start time of the h-th operation of job j is greater than the end time of the $(h-1)$-th operation of job j if $h > 1$, i.e.,

$$\hat{x}_{j,\sigma_h^j} \geq \hat{x}_{j,\sigma_{h-1}^j} + d_{j,\sigma_{h-1}^j}. \qquad (4)$$

We penalize the constraint violation by transforming it to

$$p_1 \cdot (\hat{x}_{j,\sigma_h^j} - \hat{x}_{j,\sigma_{h-1}^j} - d_{j,\sigma_{h-1}^j})^2, \qquad (5)$$

where p_1 is a sufficiently large integer. As we are minimizing the overall objective, we want (5) to evaluate to 0 when constraint (4) is satisfied and to a non-zero value proportional to its violation when it is not. Thus, we add an integer slack variable to (5) and it becomes term (3b). As long as p_1 is large enough, violating the precedence constraints leads to a large increase in the overall objective, which would reasonably be avoided by QUBO solvers.

Similarly, term (3c) is the quadratic representation of the makespan constraints. Term (3d) is the quadratic representation of the disjunctive constraints, with the help of a sufficiently large constant V to enforce the disjunction.

Time-indexed QUBO Model

The time-indexed MIP model (Kondili and Sargent 1988) is well known for its poor scalability since there is a binary variable for each ⟨job, machine, time-unit⟩ triple. However, it is straightforward to transform a time-indexed MIP model into a QUBO model, and, as we show below, there are advantages of a time-indexed QUBO (TIQ) model. In TIQ, $x_{j,m,t}$ is a binary variable that is 1 if the operation of job j on machine m starts at time t and is 0 otherwise. $\hat{C}_{max}$ is the integer makespan and $\hat{s}_j$ is the integer slack variable for

the makespan constraint of job j. The TIQ is as follows.

$$\min_{x} \ \hat{C}_{max} \tag{6a}$$

$$+ p_4 \cdot \sum_{j=1}^{N} \sum_{m=1}^{M} \left(\sum_{t=0}^{T} x_{j,m,t} - 1 \right)^2 \tag{6b}$$

$$+ p_5 \cdot \sum_{j=1}^{N} \sum_{(m_1,m_2)\in P_j} \sum_{t_1+d_{j,m_1}>t_2} x_{j,m_1t_1} \cdot x_{j,m_2t_2} \tag{6c}$$

$$+ p_6 \cdot \sum_{m=1}^{M} \sum_{j_1\neq j_2} \sum_{t_1\leq t_2<t_1+d_{j_1,m}} x_{j_1,m,t_1} \cdot x_{j_2,m,t_2} \tag{6d}$$

$$+ p_7 \cdot \sum_{j=1}^{N} \left(\sum_{t=0}^{T} \left(t \cdot x_{j,\sigma_M^j,t} \right) + d_{j,\sigma_M^j} + \hat{s}_j - \hat{C}_{max} \right)^2 \tag{6e}$$

$$\hat{C}_{max} \geq 0,\ \hat{s}_j \in \mathbb{Z}, x_{j,m,t} \in \{0,1\}, \quad \forall j = 1,...,N, \forall m = 1,...,M, \forall t = 0,...,T. \tag{6f}$$

The objective term (6a) represents the makespan to be minimized. Term (6b) ensures that an operation only starts once. T is the horizon, i.e., the upper bound of makespan and p_4, p_5, p_6 and p_7 are coefficients that penalize the value of a term if it is non-zero. Term (6c) represents the precedence constraints within a job. P_j is the set of machines pairs that have consecutive operations of job j. Term (6d) represents the resource constraints, where two jobs cannot overlap on the same machine at any time point. Term (6e) guarantees that each job completion time does not exceed the makespan.

Venturelli et al. (2016) proposed a slightly different time-indexed formulation that solves a series of feasibility problems with decreasing horizon rather than having an explicit makespan objective. However, as our purpose is to establish a baseline performance of *direct* QUBO models, we use the TIQ that is solved in a single QUBO run.

Shimada et al. (2021) experimented with a number of QUBO models for JSP on the Fujitsu Digital Annealer, with the best one being an Approximate Expression (AE) model. It is a time-indexed QUBO model that uses the sum of completion times to approximate makespan, and needs no general integer variables. The AE model replaces term (6a) by the following term, while retaining terms (6b) to (6f).

$$\sum_{j=1}^{N} \sum_{m=1}^{M} \sum_{t=0}^{T} (t + d_{j,m}) \cdot x_{j,m,t}. \tag{7}$$

Though not optimizing the makespan directly, by reducing the completion time of all operations, the AE model will tend to reduce the makespan. As far as we are aware, though the AE model is not a direct QUBO model for JSP, it is the state-of-the-art QUBO approach for JSP. We include the AE model in our experiments.

Some current specialized platforms such as the third generation of Fujitsu Digital Annealer with 100000 bits (Fujitsu Ltd. 2021) cannot accommodate JSP(400,20,100) or JSP(225,15,100) if using DQ or TIQ, respectively.[1] The D-Wave quantum annealer with 5000 qubits (D-Wave Systems Inc. 2020) can only represent smaller problems (e.g., JSP(25,5,100)) due to the smaller number and partial connectivity of qubits. Another downside of previous QUBO approaches is that they cannot handle instances with large d_{max}, e.g., $d_{max} > 10$. The limitations of the current hardware for representing and, as we show below, solving JSP with direct QUBO models motivate our large neighborhood search approach.

[1]The detailed analysis and the formal derivation for the model sizes are provided in the appendix.

Rank-Guided Large Neighborhood Search

Large Neighborhood Search (LNS) (Shaw 1998) is a framework that combines the scaling advantage of local search with the search power of specialized solvers. LNS modifies an existing solution to the problem by selecting a subset of variables and unassigning them. LNS then searches in the space induced by these unassigned variables. The choice of variables to unassign, namely the neighborhood heuristic, is crucial to the performance of LNS. For JSP, classical neighborhood heuristics identify a set of operations to focus search effort on. For example, the time window neighborhood heuristic selects all the operations that are scheduled in a particular time interval (Caseau et al. 2001). The resource load neighborhood heuristic selects operations scheduled on a particular set of resources (Carchrae and Beck 2009).

By contrast, we propose a rank-guided neighborhood heuristic that considers all operations but restricts their time windows based on rank information extracted from solutions of multiple one-machine scheduling problems. That is, the large neighborhood is formed by the heuristic restriction of variable domains, a generalization of the standard approach that fixes a subset of variable values and completely relaxes the domains of the rest.

As shown in Figure 1, the proposed LNS approach alternately solves multiple one-machine problems and a single multi-machine problem. The two main components are *rank generation* and *constrained global search*. In rank generation, one-machine problems are solved to generate operation ranks, which if combined naively might lead to cycles. In constrained global search, the multi-machine problem hence derives a time window for each operation from the single machine schedules and searches for improving, feasible solutions within the constrained time-windows. Conversely, a given global solution is used to define "rank windows" for operations in each single machine problem. The primary decision variable is the rank of each operation and the feasible domain of each rank variable is established via inference from the current global solution.

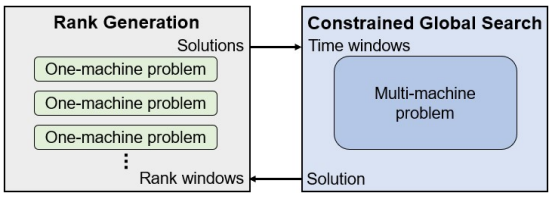

Figure 1: General description of rank-guided LNS.

Rank Generation

The rank of an operation is its position on its machine. The operation scheduled first has a rank 1, the second scheduled operation has rank 2, and so on. Ranks can serve as decision variables for JSP (Wagner 1959) since a feasible solution is obtained once all the one-machine sequences are generated and combined without cycles. Ranks can be obtained from solving one-machine scheduling problems, which have been used as subproblems in decomposition methods for JSP (e.g., the shifting bottleneck heuristic (Adams, Balas, and Zawack 1988)). Here, we obtain ranks by solving one-machine problems formulated as rank-based QUBO models.

The rank generation process is done by simultaneously solving multiple one-machine problems, one for each machine. Provided the total size of models for each machine is smaller than the hardware capacity, all models can be solved in parallel as they are independent.

The rank-based single-machine model assigns a rank to each operation on a machine, hence binary variable $x_{j,q}$ is 1 if the operation of job j is in the q-th rank and 0 otherwise. The model is as follows.

$$\min_x \sum_{q=1}^{N/2}\sum_{j=1}^{N} x_{j,q}\cdot Pos_j\cdot(N-q)\cdot d_{max} \tag{8a}$$

$$+ \sum_{q=1}^{N}\sum_{j=1}^{N} x_{j,q}\cdot(H_j-T_j)\cdot(N-q) \tag{8b}$$

$$+ p_8\cdot\sum_{j=1}^{N}\left(\sum_{q=1}^{N}x_{j,q}-1\right)^2 + p_9\cdot\sum_{q=1}^{N}\left(\sum_{j=1}^{N}x_{j,q}-1\right)^2 \tag{8c}$$

$$x_{j,q}\in\{0,1\},\quad \forall j=1,...,N,\forall q=1,...,N. \tag{8d}$$

The constant N is the number of operations on the machine and, consequently, the highest rank. In the objective, Pos_j is the precedence position of the operation j in its job. If operation j is the h-th operation in a job, then $Pos_j = h$. Recall that the job ordering is fixed for each job. Thus, for the operations that are ranked from 1 to $N/2$, the term (8a) favors small job precedence positions. Note that $N/2$ is a tunable parameter. We use $N/2$ to ensure that precedence positions are taken into consideration in the first half of the one-machine sequence.

In (8b), H_j is the 'head' (Adams, Balas, and Zawack 1988) or the inferred earliest start time of operation j and T_j is the 'tail' (Adams, Balas, and Zawack 1988) of operation j, i.e., length of the longest path from the end of the operation to a dummy node representing the makespan variable. The heads and tails are generated by constraint propagation with a makespan upper bound (see Figure 2).

Figure 2: The head and tail of operations. The dark blue boxes are the operations and the lighter blue boxes are time windows of the two operations.

The term (8b) represents the weighted sum of operation head minus tail. The weight $N-q$ prioritizes operations with small value of $(H_j - T_j)$ to start early. Terms (8c) are penalized one-hot constraints that guarantee each position of the machine sequence is assigned to only one operation and each operation is assigned only one position.

Overall, therefore, this model seeks to assign ranks such that operations with small job-precedence positions and small heads vs. tails are scheduled earlier.

Extra constraints on ranks are added to the model based on the operation time windows extracted from a feasible, partial solution to the multi-machine problem. We denote a feasible partial solution by s^p and a globally valid upper bound on makespan as UB_C. We then define the *reference time windows* as the following lower and upper bounds for operation start times obtained via constraint propagation based on s^p and UB_C:

$$[LB_{j,m}, UB_{j,m}], \forall j=1,...,N, \forall m=1,...,M. \tag{9}$$

We restrict the domain of rank variables according to the reference time windows. For the operation $o_{j,m}$ of job j on machine m, we define the following two sets:

$$B_{j,m} = \{o_{j',m} | UB_{j',m} \le LB_{j,m}\}, \tag{10a}$$

$$A_{j,m} = \{o_{j',m} | UB_{j,m} \le LB_{j',m}\}, \tag{10b}$$

where $B_{j,m}$ is the set of operations on machine m that have a maximum reference start time that is less than the minimum reference start time of $o_{j,m}$. In any improving schedule, each $o_{j',m} \in B_{j,m}$ must be scheduled before $o_{j,m}$. Similarly, $A_{j,m}$ is the set of operations that are scheduled after $o_{j,m}$ in any improving schedule. Then the resulting rank domain becomes:

$$|B_{j,m}| < r_{j,m} \le N - |A_{j,m}|, \tag{11}$$

where $r_{j,m}$ represents the rank of the operation $o_{j,m}$. In the rank-based QUBO model, we remove variables $x_{j,q}$, if $q \le |B_{j,m}|$ or $q > N - |A_{j,m}|$.

In addition, for operation $o_{j,m}$, for each operation $o_{j',m} \in B_{j,m}$, it is safe to conclude that $r_{j',m} < r_{j,m}$. Thus, we can add the following penalty term (p_{10} is a sufficiently large coefficient) to the rank-based QUBO model of machine m:

$$p_{10}\cdot \sum_{j,j'=1,\ UB_{j',m}\le LB_{j,m}}^{N} \left(\sum_{q,q'=1, q\le q'}^{N} x_{j,q}x_{j',q'}\right). \tag{12}$$

The constraints (12) are conflict constraints that can be represented as

$$\begin{aligned} &x_{j,q} + x_{j',q'} \le 1,\ \forall j,j'=1,...,N,\\ &\forall q,q'=1,...,N, q\le q',\ \text{if } UB_{j',m} \le LB_{j,m}. \end{aligned} \tag{13}$$

Note that the problem defined by (8a) to (8d) is the assignment problem that can be solved in polynomial time (Kuhn 1955). The assignment problem with conflict constraints, in the form of (13), is strongly NP-hard (Darmann et al. 2011).

By solving the QUBO models with extra constraints for all one-machine problems we obtain operation ranks that are used to constrain the multi-machine global search.

Constrained Global Search

We use a constrained version of the standard multi-machine CP model for JSP (Baptiste, Le Pape, and Nuijten 2001) to conduct the global search. In the standard CP model, operations are represented by interval variables: variables whose domain is a subset of $\{\perp\} \cup \{[s, e) | s, e \in \mathbb{Z}, s \leq e\}$, where s and e are the start and end of the interval respectively and $l = e - s$ its length. In the CP model, the interval variables $x_{j,m}$ represent each operation $o_{j,m}$, with the duration $d_{j,m} = l$, and C_{max} is the integer makespan variable.

$$\min_{x} C_{max} \tag{14a}$$

$$\text{s.t. endOf}\left(x_{j,\sigma^j_{h-1}}\right) \leq \text{startOf}\left(x_{j,\sigma^j_h}\right), \quad \forall j = 1, ..., N, \forall h = 2, ..., M, \tag{14b}$$

$$\text{endOf}\left(x_{j,\sigma^j_M}\right) \leq C_{max}, \quad \forall j = 1, ..., N, \tag{14c}$$

$$\text{disjunctive}\left(\{x_{1,m}, ..., x_{N,m}\}\right), \quad \forall m = 1, ..., M, \tag{14d}$$

$$\text{startOf}\left(x_{j,m}\right) \geq 0, \ \forall j = 1, ..., N, \forall m = 1, ..., M, \tag{14e}$$

$$C_{max} \geq 0. \tag{14f}$$

The objective function is to minimize makespan, as stated in (14a). Constraints (14b) are the precedence constraints which guarantee that all operations of a job are processed in order. Constraints (14c) ensure that the makespan is at least the largest completion time of the last operation of all jobs. Constraints (14d) are global constraints which ensure that no two operations overlap on a given machine.

Given a set of one-machine schedules (e.g., from the rank generation procedure) along with a feasible solution of the multi-machine CP model, we add the following constraints to the CP model.

We define an integer parameter k that controls the size of the neighborhood. For machine m, we define the given operation ranks as *reference ranks*, denoted by $\{1 \leq g_{j,m} \leq N, \forall j = 1, ..., N\}$. For machine m, job j_1 and job j_2, if the condition $g_{j_1,m} + k \leq g_{j_2,m} - k$ is satisfied, then $o_{j_1,m}$ executes before $o_{j_2,m}$. Thus, we add the following constraints:

$$\text{endOf}(x_{j_1,m}) \leq \text{startOf}(x_{j_2,m}), \ \forall m = 1, ..., M, \quad \forall j_1, j_2 = 1, ..., N, \text{ if } g_{j_1,m} + k \leq g_{j_2,m} - k. \tag{15}$$

We also constrain the time windows of each operation based on reference ranks. We sort the durations of operations on the same machine in ascending order to get the following ordered set for each machine m:

$$\{d^1_m, d^2_m, ..., d^N_m\}, \tag{16}$$

where $d^1_m \leq d^2_m \leq ... \leq d^{N-1}_m \leq d^N_m$. To enforce a rank window of $[g_{j,m} - k, g_{j,m} + k]$ to the operation $o_{j,m}$, then we add the following constraints:

$$\sum_{i=1}^{g_{j,m}-k-1} d^i_m \leq \text{startOf}(x_{j,m}) \leq C_{max} - \sum_{i=1}^{N-g_{j,m}-k+1} d^i_m \quad \forall j = 1, ..., N, \forall m = 1, ..., M. \tag{17}$$

The *rank-guided neighborhood* is defined by model (14) and constraints (15) and (17). The size of the neighborhood can be tuned by the parameter k. With a small k (e.g., $k = 1$), an exact search algorithm can quickly find better solutions or conclude that none exist in the neighborhood. A larger k increases both the search effort and the likelihood that the neighborhood contains an improving solution.

Rank-Guided LNS

The full rank-guided LNS procedure is shown in Algorithm 1. At line 1, the best solution sol^* and the best objective value v^* are set to sol_0 and v_0. Until the time limit, the algorithm repeats lines 3 to 11.

Algorithm 1: Rank-guided LNS

input : Data of a JSP instance; an initial solution/objective sol_0/v_0; an initial runtime for constrained global search t_0; the time limit T_{end}, an initial k_0.

output : The best solution/objective sol^*/v^*.

```
1  v* ← v_0, sol* ← sol_0;
2  while Runtime ≤ T_end do
3      (M_r, M_u) ← RelaxMachSeq(sol*,ratio);
4      tw_r ← Propagation(sol*,v*,M_r,M_u);
5      rw_r ← RankGeneration(M_r, tw_r);
6      k ← k_0, status ← Infeasible, t_s ← t_0;
7      while status ≠ Feasible and k < N/3 do
8          (sol, v, status) ← CGSearch(rw_r, sol*, M_u,
             k, t_s);
9          (k, t_s) ← ktUpdate(status);
10         if v < v* then
11             sol* ← sol, v* ← v;
```

At line 3, a number of machines depending on a $ratio$ are randomly selected and their operation sequences are relaxed. M_r and M_u are the sets of relaxed and unrelaxed machines and $ratio = M_r/(M_r + M_u)$. The algorithm then performs constraint propagation with the best-known objective value and the precedence graph induced by the relaxed and unrelaxed machines, producing time windows tw_r for the operations on the relaxed machines. At line 5, the QUBO models of the one-machine problems are solved in parallel.

The initialization of constrained global search is performed at line 6. While there is no feasible solution and k has not reached its limit at line 8, the algorithm repeatedly searches for better solutions with increasing k values. Note that when k reaches $N/2$, the constrained CP model would be the same as the original CP model. Thus, the limit of k is set to a smaller value $N/3$. t_s is the running time of the constrained global search which is updated according to the solution status, as detailed in the next section.

In the search, constraints (15) and (17) are added to operations on relaxed and unrelaxed machines. The reference ranks of the operations on M_r are based on the solving one-machine problems at line 5, while the reference ranks of the operations on M_u are the corresponding operation ranks in the best feasible solution sol^*. As a consequence, operations on M_u also have time windows constrained by (15) and (17).

The k and t_s are updated according to the solver status, as shown at line 9. Thus, the growing neighborhood mechanism (Carchrae and Beck 2009) has been realized by increasing k gradually: operations get less constrained as the neighborhood grows. At line 10 and 11, when feasible solutions are found, if the best solution among them is better than sol^*, the algorithm replaces sol^* and v^* for the next iteration. Otherwise, it keeps using the current solution.

Empirical Evaluation

In this section, we present our experimental results. Recall that our goal is to investigate if specialized hardware can be used to solve non-trivial JSP problem instances.

Our solvers are the CP solver IBM ILOG CP Optimizer (CPO) V20.1.0, the MIP solver Gurobi V9.5.0, and the QUBO solver, the Fujitsu Digital Annealer (DA) (Matsubara et al. 2020). The third generation DA (DA3) is a hybrid system of hardware and software with a capacity of 100,000 binary variables. For our DA environment,[2] the coefficients for the quadratic terms range from -2^{62} to 2^{62} and those for the linear terms range from -2^{73} to 2^{73} (Fujitsu Ltd. 2021). Different from the quantum annealing algorithms, the DA algorithm is a classical algorithm based on simulated annealing (SA), that takes advantage of the massive parallelization provided by the custom CMOS hardware. The difference between the SA and DA algorithm are as follows:

- DA utilizes parallel tempering that runs a number of problem solving processes (*replicas*) in parallel with different temperatures. Replicas can swap temperatures to diversify the search (Dabiri et al. 2020). In a replica, each Monte Carlo step considers all one-bit flips in parallel.
- DA supports a dedicated bit flip mechanism, over a subset of variables belonging to one-hot equivalent constraints when using DA3.
- DA can deal with inequality constraints that are not modeled in QUBO.

For our experiments, we run DA3 on a remote computer and do not include the communication time in our runtime limits and results. The programs for running DA3, CPO, and Gurobi are written in Python 3.7 and run on a Ubuntu computer with Intel(R) Core(TM) i7-9700K CPU @3.00GHz with 32 GB RAM.

Experiment Metrics

We use the best objective value found and the mean relative error as two measures. Let $B_{i,t,a}$ be the best solution attained by runtime t of approach a for instance i. The relative error at time t for approach a on instance i is shown in (18) where B_i represents the best known *lower bound* in the literature for known benchmarks or the best *feasible objective value* over all approaches for generated instances. For a minimization problem, (18) is always non-negative. The mean relative error of approach a at time t, $MRE(t,a)$, is computed in (19).

$$RE(i,t,a) = \frac{(B_{i,t,a} - B_i)}{B_i} \tag{18}$$

$$MRE(t,a) = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} RE(i,t,a) \tag{19}$$

Experiments on Small Instances

As an initial experiment, we restrict ourselves to small problems to evaluate the direct models to test if direct models are promising enough to try on large problem instances as future hardware capacity evolves.

We test 5 randomly generated JSP(25,5,100) instances and 5 randomly generated JSP(100,10,100) instances, with integer operation processing times uniformly drawn from $[1..100]$. We obtain initial solutions and associated makespans by using the shortest-processing time (SPT) dispatching rule (Conway 1965). Then, all the JSP instances are preprocessed by constraint propagation to generate lower and upper bounds on the makespan and on the start time of each operation. The QUBO sizes of direct models are all significantly smaller than a naive model.

For the DA3, we use the default parameter configuration that works reasonably well across the instances. The penalty terms are selected so there is no violation to any constraint. The DA3 was run for 20s for direct models and 1s for rank-based models. The total runtime of the rank-guided LNS is 20s for fair comparison. When testing TIQ with approximate expression (TIQAE) (Shimada, Shibuya, and Shibasaki 2021), DA3 was run for 1s per iteration and 20 iterations (20s in total), matching the authors' parameterization.

For the rank-guided LNS, the constrained global search is initialized by setting $ratio$ to 0.7 and k_0 to 1. Once the solver status is 'Infeasible' or 'Timeout', k is increased by $\lceil N/10 \rceil$, where N is the number of jobs. The runtime of the multi-machine CP model on CPO is fixed to 1s ($t_s = 1$).

For comparison, we run the standard CP model defined in (14) and classical CP-based LNS with the same CP model using CPO. We select the time window neighborhood with two windows and the resource load neighborhood with 65% resources (Carchrae and Beck 2009). We also run a disjunctive MIP model (Manne 1960) on Gurobi.

The total runtime of each of the aforementioned approaches is 20s. CPO and Gurobi are set to the default configurations. The makespan results are shown in Table 1.

In the table, RGLNS is the rank-guided LNS. TW is the time window neighborhood and RL is the resource load neighborhood. The '*' indicates proved optimality. For both 5×5 and 10×10 instances, the standard CP model and the disjunctive MIP model find and prove an optimal solution in 2s. RGLNS is competitive as it finds the same solutions in 2s, but, as a heuristic method, is unable to prove optimality. The classical neighborhood TW is better than RL, but

	RGLNS	CP-LNS		Existing	Direct model			
Size	QUBO/CP	TW	RL	TIQAE	DQ	TIQ	MIP	CP
5	**355**	387	387	**355**	inf	**355**	***355**	***355**
×	**282**	**282**	**282**	**282**	inf	**282**	***282**	***282**
5	**372**	**372**	**372**	**372**	inf	**372**	***372**	***372**
	369	**369**	372	**369**	inf	**369**	***369**	***369**
	355	**355**	**355**	**355**	inf	**355**	***355**	***355**
10	**780**	818	917	inf	inf	inf	***780**	***780**
×	**788**	802	838	inf	inf	inf	***788**	***788**
10	**872**	882	920	inf	inf	inf	***872**	***872**
	785	787	866	inf	inf	inf	***785**	***785**
	766	775	821	inf	inf	inf	***766**	***766**

Table 1: Makespan of small instances.

[2]Experiments were conducted on the Digital Annealer environment prepared exclusively for research at the University of Toronto.

both are worse than RGLNS. The TIQ and TIQAE achieve good results for 5×5 instances but cannot find any feasible solutions for 10×10 instances. The DQ is the worst as it produces no feasible solutions to any instance.

The results demonstrate that DA3 can solve very small JSP instances with TIQ or TIQAE but the solution quality degrades substantially with even modest increases in problem size. Although 10×10 JSP is not the largest problem DA3 can represent with DQ or TIQ, our experiments already show the challenge in terms of directly solving JSP.

Experiments on Large Instances

We test JSP(400,20,100) Taillard benchmark instances from the OR-Library (Beasley 1990). Direct QUBO models for these problems are too large for DA3 and hence are not included. As on the small instances, DA3 is run 1s for rank generation. For the constrained global search running on CPO in RGLNS, we still set $ratio$ to 0.7 and $k_0 = 1$ and update k by $\lceil N/10 \rceil$ once the CPO status is 'Infeasible' or 'Timeout'. We set $t_0 = 5$ as we only want to put little effort on proving infeasibility when k is small. Once the solution status is 'Timeout', the CPO runtime is updated to larger values: $10N$ seconds, as shown at line 9 of Algorithm 1.

As shown in Table 2, rank-guided LNS achieves good results. CPO running for $10N$s finds five best solutions compared to the other approaches. The rank-guided LNS is much better than the disjunctive MIP model and the classical LNS with time window or resource load neighborhoods.

The MRE graph comparing against best-known lower bounds is shown in Figure 3. At the bottom, we can see that RGLNS with DA3 and CPO running for 1s and $10N$s performs almost identically to the standard CP model.

Scaling to Larger Instances

To evaluate the limit of the RGLNS, we do further experiments on randomly generated square JSP instances of size $\{25, 30, 35, 40, 45, 50\}$, with operation duration uniformly drawn between 1 and 100, as there is no such benchmark instances in the literature. We focus on square instances as they are in general more difficult than rectangular instances (Watson et al. 2003). The standard CP model, the disjunctive MIP model, RGLNS, and the two classical LNS are run for 3600s per instance for 10 instances per size. The configurations of DA3/CPO/Gurobi and the parameters of RGLNS remain the same as the experiments on Taillard instances. The final MRE comparing against the best feasible objective values over all approaches with respect to problem sizes is plotted in Figure 4.

		RGLNS	CP-LNS		Direct model	
Taillard		QUBO/CP	TW	RL	MIP	CP
20×20	21	1659	1727	1748	1684	**1655**
	22	**1620**	1640	1689	1641	1629
	23	**1569**	1604	1629	1588	1584
	24	1652	1703	1745	1718	***1644**
	25	1601	1619	1719	1638	**1598**
	26	**1660**	1679	1704	1711	1674
	27	1699	1749	1781	1723	**1693**
	28	1604	1648	1714	1633	***1603**
	29	**1628**	1653	1679	1651	1634
	30	**1598**	1619	1669	1636	1599

Table 2: Makespan of selected Taillard instances.

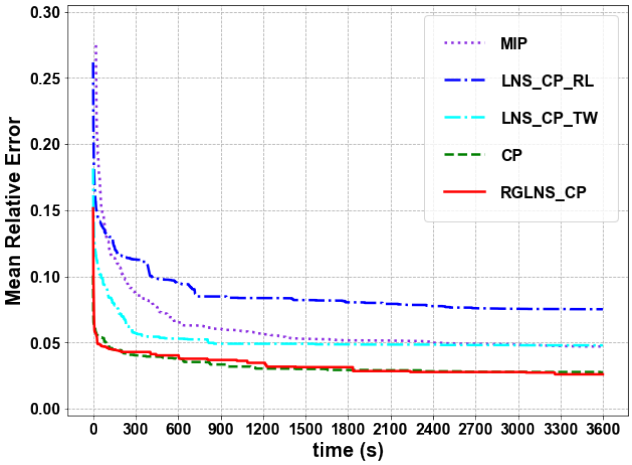

Figure 3: MRE of 20×20 Taillard instances.

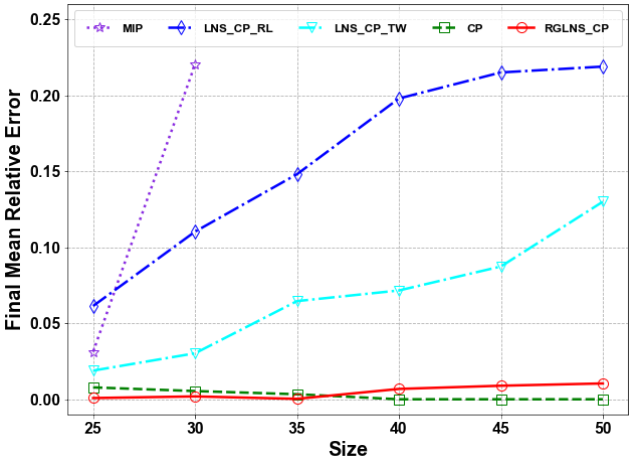

Figure 4: Final MRE of square JSP instances.

The figure shows that the RGLNS is on average better than direct CP when problem size is smaller than 40 but underperforms CP after 40. The performance of MIP and classical LNS deteriorates rapidly as the problem size increases. In particular, Gurobi with the disjunctive MIP model cannot find any feasible solution for 35×35 or larger instances in 3600s. By contrast, RGLNS has a much slighter relative performance degradation with increasing problem size.

The MRE plot of the 60 instances over time is shown in Figure 5. For the instances that Gurobi fails to find any solution, we use the SPT makespan to calculate the MRE. We can see that RGLNS significantly outperforms MIP and classical CP-based LNS, while being competitive with CP.

Comparing One-Machine Models

The one-machine scheduling problems of the rank-guided LNS can be solved by other methods. A key question is

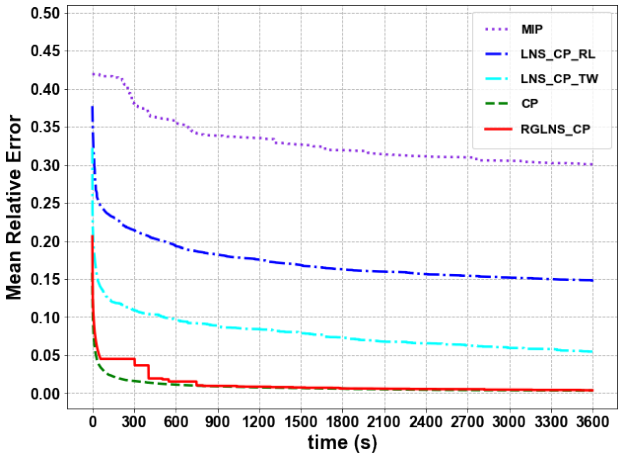

Figure 5: MRE of square JSP instances.

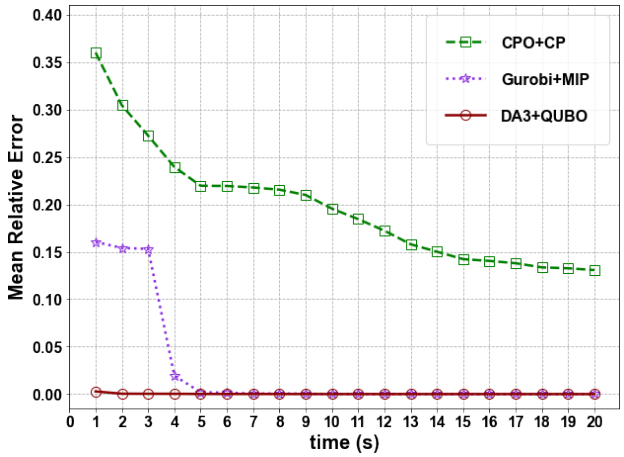

Figure 6: MRE of one-machine scheduling problems.

whether the use of the DA3 provides advantages over alternative techniques within the RGLNS approach. Hence, we compare DA3 with the QUBO model defined above, CP Optimizer (CPO) with a rank-based CP model, and Gurobi with a rank-based MIP model. While having the same decision variables $x_{j,q}$ as the rank-based QUBO model, the MIP/CP model also use (8a) + (8b) as the objective function, with variable pruning induced by (11). Instead of terms (8c) and (12), the MIP/CP models have constraints (13) and the following constraints:

$$\sum_{j=1}^{N} x_{j,q} = 1, \forall q = 1, ..., N. \\ \sum_{q=1}^{N} x_{j,q} = 1, \forall j = 1, ..., N. \tag{20}$$

We randomly generate 10 instances each for one-machine problems with the operation number in {10,20,30,40,50,60, 70,80} and the operation heads and tails uniformly drawn between 1 and 500. The window of each operation start time is given by constraint propagation with some fixed machine sequences obtained from the SPT solution. The three methods are each run for 5s. The MRE results are shown in Figure 6. DA3 and Gurobi reach similar solution qualities at 20s, while DA3 finds good solutions much faster. The CPO performance is consistently worse than Gurobi and DA3 throughout the 20s.

These results show that, for the one-machine problems in RGLNS, the third generation Fujitsu Digital Annealer outperforms existing state-of-the-art approaches, demonstrating its value within the RGLNS framework.

Discussion

The better performance of TIQAE than TIQ and the failure of DQ cast a shadow on using binary expansion for integer variables in QUBO. We believe that the DA3 does not perform well with the binary representation of integer variables due to its one-flip neighborhood. Flipping one bit in a binary representation of an integer can lead to very distant integers, requiring changes to many other binary variables to achieve an energy of zero (i.e., feasibility). Since general integers naturally appear in many optimization problems, an important question for future work is whether difficulties in solving problems with integer variables are seen with other QUBO hardware.

Our hybrid approach that incorporates traditional optimization techniques and novel hardware makes solving large JSP with a QUBO-based approach possible. The largest JSP that previous QUBO-based methods could solve on specialized hardware is a 6×6 instance with tiny $d_{max} = 2$ (Venturelli, Marchand, and Rojo 2016). Our approach can solve 20×20 to 50×50 JSPs with $d_{max} = 100$ and, as shown by our experiments, achieves results that are competitive with state-of-the-art constraint programming based approaches. Note that for Taillard 20×20 benchmark instances, the makespans found by RGLNS are on average 0.7% worse than the makespans found by state-of-the-art metaheuristics (Shylo 2020). Our goal, however, was not to compete with JSP-specific metaheuristic techniques but to investigate if non-trivial scheduling problems can be solved using hardware designed for a general optimization problem.

Conclusion

Our results show that current specialized hardware for solving Quadratic Unconstrained Binary Optimization problems can only represent small job shop scheduling problems when using a direct encoding and further that even on small instances, direct models perform poorly.

By contrast, we showed for the first time that it is possible to employ such specialized hardware in combination with constraint programming to solve large-scale job shop scheduling problems. The hybrid performance is substantially better than mixed integer programming and competitive with a state-of-the-art constraint programming model.

Acknowledgements The authors would like to thank Fujitsu Ltd. and Fujitsu Consulting (Canada) Inc. for providing financial support and access to the Digital Annealer at the University of Toronto. Partial funding for this work was provided by Fujitsu Ltd. and the Natural Sciences and Engineering Research Council of Canada.

References

Adams, J.; Balas, E.; and Zawack, D. 1988. The shifting bottleneck procedure for job shop scheduling. *Management science*, 34(3): 391–401.

Aramon, M.; Rosenberg, G.; Valiante, E.; Miyazawa, T.; Tamura, H.; and Katzgraber, H. G. 2019. Physics-inspired optimization for quadratic unconstrained problems using a digital annealer. *Frontiers in Physics*, 7: 48.

Baptiste, P.; Le Pape, C.; and Nuijten, W. 2001. *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer Science & Business Media.

Beasley, J. E. 1990. OR-Library: distributing test problems by electronic mail. *Journal of the operational research society*, 41(11): 1069–1072.

Carchrae, T.; and Beck, J. C. 2009. Principles for the design of large neighborhood search. *Journal of Mathematical Modelling and Algorithms*, 8(3): 245–270.

Caseau, Y.; Laburthe, F.; Le Pape, C.; and Rottembourg, B. 2001. Combining local and global search in a constraint programming environment. *The Knowledge Engineering Review*, 16(1): 41–68.

Coffrin, C.; Nagarajan, H.; and Bent, R. 2019. Evaluating ising processing units with integer programming. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 163–181. Springer.

Cohen, E.; Senderovich, A.; and Beck, J. C. 2020. An ising framework for constrained clustering on special purpose hardware. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 130–147. Springer.

Conway, R. W. 1965. Priority dispatching and job lateness in a job shop. *J. Ind. Eng.*, 16(4): 228–237.

D-Wave Systems Inc. 2020. Hybrid Solver for Discrete Quadratic Models. https://www.dwavesys.com/sites/default/files/14-1050A-A_Hybrid_Solver_for_Discrete_Models.pdf. Accessed: 2022-03-01.

Dabiri, K.; Malekmohammadi, M.; Sheikholeslami, A.; and Tamura, H. 2020. Replica exchange mcmc hardware with automatic temperature selection and parallel trial. *IEEE Transactions on Parallel and Distributed Systems*, 31(7): 1681–1692.

Darmann, A.; Pferschy, U.; Schauer, J.; and Woeginger, G. J. 2011. Paths, trees and matchings under disjunctive constraints. *Discrete Applied Mathematics*, 159(16): 1726–1735.

Fujitsu Ltd. 2021. The third generation of the Digital Annealer. https://www.fujitsu.com/jp/group/labs/en/documents/about/resources/tech/techintro/3rd-g-da_en.pdf. Accessed: 2022-03-01.

Kochenberger, G.; Hao, J.-K.; Glover, F.; Lewis, M.; Lü, Z.; Wang, H.; and Wang, Y. 2014. The unconstrained binary quadratic programming problem: a survey. *Journal of combinatorial optimization*, 28(1): 58–81.

Kondili, E.; and Sargent, R. 1988. *A general algorithm for scheduling batch operations*. Department of Chemical Engineering, Imperial College.

Kuhn, H. W. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2): 83–97.

Kurowski, K.; Weglarz, J.; Subocz, M.; Różycki, R.; and Waligóra, G. 2020. Hybrid Quantum Annealing Heuristic Method for Solving Job Shop Scheduling Problem. In *International Conference on Computational Science*, 502–515. Springer.

Lawler, E.; Lenstra, J.; Kan, A.; and Shmoys, D. 1993. Sequencing and scheduling: Algorithms and complexity. *Handbooks in operations research and management science*, 4: 445–522.

Manne, A. 1960. On the job-shop scheduling problem. *Operations Research*, 8(2): 219–223.

Matsubara, S.; Takatsu, M.; Miyazawa, T.; Shibasaki, T.; Watanabe, Y.; Takemoto, K.; and Tamura, H. 2020. Digital annealer for high-speed solving of combinatorial optimization problems and its applications. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 667–672. IEEE.

Mohseni, M.; Read, P.; Neven, H.; Boixo, S.; Denchev, V.; Babbush, R.; Fowler, A.; Smelyanskiy, V.; and Martinis, J. 2017. Commercialize quantum technologies in five years. *Nature News*, 543(7644): 171.

Nowicki, E.; and Smutnicki, C. 2005. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8(2): 145–159.

Shaw, P. 1998. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, 417–431. Springer.

Shaydulin, R.; Ushijima-Mwesigwa, H.; Safro, I.; Mniszewski, S.; and Alexeev, Y. 2019. Network community detection on small quantum computers. *Advanced Quantum Technologies*, 2(9): 1900029.

Shimada, D.; Shibuya, T.; and Shibasaki, T. 2021. A Decomposition Method for Makespan Minimization in Job-Shop Scheduling Problem Using Ising Machine. In *IEEE 8th International Conference on Industrial Engineering and Applications (ICIEA 2021)*. IEEE.

Shylo, O. 2020. Best known lower and upper bounds of Taillard instances. http://optimizizer.com/TA.php. Accessed: 2022-03-01.

Venturelli, D.; Marchand, D.; and Rojo, G. 2016. Job shop scheduling solver based on quantum annealing. In *Proc. of ICAPS-16 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling (COPLAS)*, 25–34.

Wagner, H. 1959. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6(2): 131–140.

Watson, J.; Beck, J. C.; Howe, A.; and Whitley, L. 2003. Problem difficulty for tabu search in job-shop scheduling. *Artificial intelligence*, 143(2): 189–217.