Job Shop Scheduling Solver based on Quantum Annealing Supplemental Material

Abstract

In this supplemental material document, we provide further details on the problem compilation and characterization techniques as well as the classical algorithms used for our study of the job-shop scheduling problem (JSP). We review minor-embedding, present our partitioning of the energy sector for the timespan discrimination method, and explain how to optimize the binary search given the information extracted from a pre-characterization procedure. The classical algorithms used for variable pruning and for benchmarking are also discussed.

JSP and QUBO formulation

In order to make this document more self-contained and convenient, we first summarize the notation and quadratic unconstrained binary optimization (QUBO) formulation introduced in our paper on the job-shop scheduling problem (JSP) before expanding on the penalty form used for the constraints and the timespan discrimination scheme.

Notation and formulation

We consider a JSP for a set of M machines $\{\mathbf{m}_m\}$ and N jobs $\mathbf{j}_n$, each composed of an ordered sequence of operations. Operations $\{O_i\}$ are enumerated in a lexicographical order starting with the first job $\mathbf{j}_1$ and finishing with the last job $\mathbf{j}_N$ such that

$$\begin{aligned} \mathbf{j}_1 &= \{O_1 \to \cdots \to O_{k_1}\}, \\ \mathbf{j}_2 &= \{O_{k_1+1} \to \cdots \to O_{k_2}\}, \\ &\cdots \\ \mathbf{j}_N &= \{O_{k_{N-1}+1} \to \cdots \to O_{k_N}\}, \end{aligned} \tag{1}$$

with $i \in \{1, \ldots, k_N\}$ being the index running over all operations. We let q_i be the index of the machine $\mathbf{m}_{q_i}$ responsible for executing operation O_i, and p_i the execution time of the operation. We note that execution times are often assumed to be larger than zero in the literature. A simple generalization, which allows for the addition of extra constraints without changing the problem description, consists in allowing execution times of zero. This does not contribute to the length of a job or the makespan directly, but $p_i = 0$ introduces the extra constraint that machine $\mathbf{m}_{q_i}$ should be either idle when this operation is scheduled or about to start another operation at this very point in time.

Our QUBO formulation for the decision version of the problem assigns a set of binary variables for each operation, corresponding to the various possible discrete starting times the operation can have:

$$x_{i,t} = \begin{cases} 1 & : \text{ operation } O_i \text{ starts at time } t, \\ 0 & : \text{ otherwise.} \end{cases} \tag{2}$$

Here t is bounded from above by the timespan T, which represents the maximum time we allow for the jobs to complete.

The classical Hamiltonian,

$$H_T(\bar{x}) = \eta h_1(\bar{x}) + \alpha h_2(\bar{x}) + \beta h_3(\bar{x}), \tag{3}$$

is composed of 3 constraints encoded as penalty terms given by

$$h_1(\bar{x}) = \sum_n \left(\sum_{\substack{k_{n-1}<i<k_n \\ t+p_i>t'}} x_{i,t} x_{i+1,t'} \right), \tag{4}$$

$$h_2(\bar{x}) = \sum_m \left(\sum_{(i,t,k,t') \in R_m} x_{i,t} x_{k,t'} \right), \tag{5}$$

$$h_3(\bar{x}) = \sum_i \left(\sum_t x_{i,t} - 1 \right)^2, \tag{6}$$

where $R_m = A_m \cup B_m$ and

$$\begin{aligned} A_m &= \{(i,t,k,t') : (i,k) \in I_m \times I_m, \\ &\quad i \neq k, 0 \leq t, t' \leq T, 0 < t' - t < p_i\}, \\ B_m &= \{(i,t,k,t') : (i,k) \in I_m \times I_m, \\ &\quad i < k, t' = t, p_i > 0, p_j > 0\}. \end{aligned}$$

Penalties versus rewards formulation

The encoding of constraints as terms in a QUBO problem can either reward the respecting of these constraints or penalize their violation. Although the distinction may at first

seem artificial, the actual QUBO problem generated differs and can lead to different performance on an imperfect annealer. We present one such alternative formulation where the precedence constraint (4) is instead encoded as a reward for correct ordering by replacing $+\eta h_1(\bar{x})$ with $-\eta' h_1'(\bar{x})$, where

$$h_1'(\bar{x}) = \sum_n \left(\sum_{\substack{k_{n-1}<i<k_n \\ t+p_i \leq t'}} x_{i,t} x_{i+1,t'} \right). \tag{7}$$

The new Hamiltonian is

$$H_T'(\bar{x}) = -\eta' h_1'(\bar{x}) + \alpha h_2(\bar{x}) + \beta h_3(\bar{x}). \tag{8}$$

The reward attributed to a solution is equal to η' times the number of satisfied precedence constraints. A feasible solution, where all constraints are satisfied, will have energy equal to $-\eta'(k_N - N)$.

The functions h_1 and h_1' differ only by the range of t': in the rewards version we have

$$t' - t \geq p_i,$$

and in the penalties version we have

$$t' - t < p_i.$$

The fact that we are allowing equality in the rewards version means that h_1' will always have more quadratic terms than h_1 regardless of variable pruning, leading to a more connected QUBO graph and therefore a harder problem to embed.

Another important disadvantage is revealed when choosing the coefficients η', α, and β in H_T' to guarantee that no unfeasible solution has energy less than $-\eta'(k_N - N)$. This can happen if the penalty that we gain from breaking constraints h_2 or h_3 is less than the potential reward we get from h_1'. The penalty-based formulation simply requires that η, α, and β be larger than 0. The following lemma summarizes the equivalent condition for the reward-based case.

Lemma 1. If $\beta/\eta' \geq 3$ and $\alpha > 0$, then

$$H_T'(\bar{x}) \geq -(k_N - N), \tag{9}$$

for all $\bar{x}$, with equality if and only if $\bar{x}$ represents a feasible schedule.

We also found examples that show that these bounds on the coefficients are tight.

The fact that β/η' must be greater than or equal to 3 is a clear disadvantage because of the issues with precision of the current hardware. In H_T we can set all of the penalty coefficients (and hence all non-zero couplers) to be equal, which is the best possible case from the point of view of precision.

Timespan discrimination

The timespan discrimination that we propose is a specification to strike a compromise between the information obtained from each solver call, and the required precision for this information to be accurate and obtained efficiently. Specifically, we want this extra information to help identify the optimal makespan by looking at the energy of the solutions. This means breaking the degeneracy of the ground states (i.e., the valid solutions) and assigning different energy sectors to different makespans. To prevent collisions with invalid solutions, these energy sectors have to fit within the logical QUBO problem's gap given by $\Delta E = \min\{\eta, \alpha, \beta\}$. We note that this will affect the actual gap (as seen by the hardware) of the embedded Ising model.

Since the binary variables we have defined in the proposed formulation are not sufficient to write a simple expression for the makespan of a given solution, additional auxiliary variables and associated constraints would need to be introduced. Instead, a simple way to implement this feature in our QUBO formulation is to add a number of local fields to the binary variables corresponding to the last operation of each job, $\{O_{k_1}, \ldots, O_{k_N}\}$. Since the makespan depends on the completion time of the last operation, the precedence constraint guaranties that the makespan of a valid solution will be equal to the completion time of one of those operations. We can then select the local field appropriately as a function of the time index t to penalize a fixed number K of the larger makespans ranging from $T-K+1$ to T. Within a sector assigned to the time step $\mathcal{T}$, we need to further divide $\Delta E_{\mathcal{T}}$ by the maximum number of operations that can complete at $\mathcal{T}$ to obtain the largest value we can use as the local field $h_{\mathcal{T}}$, i.e., the number of distinct machines used by at least one operation in the set of operations $\{O_{k_1}, \ldots, O_{k_N}\}$, denoted by M_{final}. If K is larger than 1, we also need to ensure that contributions from various sectors can be differentiated. The objective is to assign a distinct T-dependent energy range to all valid schedules with makespans within $[T-K, T]$. More precisely, we relate the local fields for various sectors with the recursive relation

$$h_{\mathcal{T}-1} = \frac{h_{\mathcal{T}}}{M_{\text{final}}} + \epsilon, \tag{10}$$

where ϵ is the minimum logical energy resolvable by the annealer. Considering that ϵ is also the minimum local field we can use for $h_{\mathcal{T}-K+1}$ and that the maximum total penalty we can assign through this time-discrimination procedure is $\Delta E - \epsilon$, it is easy to see that the energy resolution should scale as $\Delta E/(M_{\text{final}}^K)$. An example of the use of local fields for timespan discrimination is shown in Figure 1-d of the main text for the case $K = 1$.

Computational strategy

This section elaborates on the compilation methods and the quantum annealing implementation of a full optimization solver based on the decision version and a binary search as outlined in the main text.

Compilation

The process of compiling, or *embedding*, an instance for a specific target architecture is a crucial step given the locality of the programmable interactions on current quantum annealer architectures. During the graph-minor topological embedding, each vertex of the problem graph is mapped to a subset of connected vertices, or subgraph, of the hardware graph. These assignments must be such that the the edges in

the problem graph have at least one corresponding edge between the associated subgraphs in the hardware graph. Formally, the classical Hamiltonian Eq. (3) is mapped to a quantum annealing Ising Hamiltonian on the hardware graph using the set of equations that follows. The spin operators $s\vec{\sigma}_i$ are defined by setting $s = 1$ and using the Pauli matrices to write $\vec{\sigma}_i = (\sigma_i^x, \sigma_i^y, \sigma_i^z)$. The resulting spin variables $\sigma_i^z = \pm 1$, our qubits, are easily converted to the usual binary variables $x_i = 0, 1$ with $\sigma_i^z = 2x_i - 1$. The Ising Hamiltonian is given by

$$H = A(t)\left[H_Q + H_E\right] + B(t)H_D, \tag{11}$$

$$H_Q = \sum_{ij} J_{ij}\sigma_{\alpha_i}^z \sigma_{\beta_j}^z\big|_{(\alpha_i,\beta_j)\in E(i,j)} + \sum_{\substack{i \\ k\in V(i)}} \frac{h_i}{N_{V(i)}}\sigma_k^z \tag{12}$$

$$H_E = -\sum_{\substack{i \\ (k,k')\in E(i,i)}} J_{i,k,k'}^F \sigma_k^z \sigma_{k'}^z, \tag{13}$$

$$H_D = \sum_{\substack{i \\ k\in V(i)}} \sigma_k^x, \tag{14}$$

where for each logical variable index i we have a corresponding ensemble of qubits given by the set of vertices $V(i)$ in the hardware graph with $|V(i)| = N_{V(i)}$. Edges between logical variables are denoted $E(i,j)$ and edges within the subgraph of $V(i)$ are denoted $E(i,i)$. The couplings J_{ij} and local fields h_i represent the logical terms obtained after applying the linear QUBO-Ising transformation to Eq. (3). $J_{i,k,k'}^F$ are embedding parameters for vertex $V(i)$ and $(k,k') \in E(i,i)$ (see discussion below on the ferromagnetic coupling). The equation above assumes that a local field h_i is distributed uniformly between the vertices $V(i)$ and the coupling $J_{i,j}$ is attributed to a single randomly selected edge (α_i, β_j) among the available couplers $E(i,j)$, but other distributions can be chosen. In the actual hardware implementation we rescale the Hamiltonian by dividing by J_F, which is the value assigned to all $J_{i,k,k'}^F$, as explained below. This is due to the limited range of J_{ij} and h_i allowed by the machine Venturelli et al. (2015).

Once a valid embedding is chosen, the ferromagnetic interactions $J_{i,k,k'}^F$ in Eq. (13) need to be set appropriately. While the purpose of these couplings is to penalize states for which $\langle\sigma_k^z\rangle \neq \langle\sigma_{k'}^z\rangle$ for $k, k' \in V(i)$, setting them to a large value negatively affects the performance of the annealer due to the finite energy resolution of the machine (given that all parameters must later be rescaled to the actual limited parameter range of the solver) and the slowing down of the dynamics of the quantum system associated with the introduction of small energy gaps. There is guidance from research in physics (Venturelli et al. (2015); King, Lanting, and Harris (2015)) and mathematics (Choi (2008)) on which values could represent the optimal $J_{i,k,k'}^F$ settings, but for application problems it is customary to employ empirical prescriptions based on pre-characterization (Rieffel et al. (2015)) or estimation techniques of performance (Perdomo-Ortiz et al. (2015)).

Despite embedding being a time-consuming classical computational procedure, it is usually not considered part of the computation and its runtime is not measured in determining algorithmic complexity. This is because we can assume that for parametrized families of problems one could create and make available a portfolio of embeddings that are compatible with all instances belonging to a given family. The existence of a such a library would reduce the computational cost to a simple query in a lookup table, but this could come at the price of the available embedding not being fully optimized for the particular problem instance.

Quantum annealing optimization solver

We detail an approach that can be used to solve individual JSP instances. We shall assume the instance at hand can be identified as belonging to a pre-characterized family of instances for minimal computational cost. This can involve identifying N, M, and θ, as well as the approximate distribution of execution times for the operations. The pre-characterization is assumed to include a statistical distribution of optimal makespans as well as the appropriate solver parameters ($J_{\rm F}$, optimal annealing time, etc.). Using this information, we need to build an ensemble of queries $\mathcal{Q} = \{\mathtt{q}\}$ to be submitted to the D-Wave quantum annealer to solve a problem H. Each element of $\mathcal{Q}$ is a triple (t_A, R, T) indicating that the query considers R identical annealings of the embedded Hamiltonian H_T for a single annealing time t_A. To determine the elements in $\mathcal{Q}$ we first make some assumptions, namely, i) *sufficient statistics*: for each query, R is sufficiently large to sample appropriately the ensembles defined in Eqs. (17)–(19); ii) *generalized adiabaticity*: t_A is optimized (over the window of available annealing times) for the best annealing performance in finding a ground state of H_T (i.e., the annealing procedure is such that the total expected annealing time $t_A R^\star$ required to evolve to a ground state is as small as possible compared to the time required to evolve to an excited state, with the same probability). Both of these conditions can be achieved in principle by measuring the appropriate optimal parameters $R^\star(\mathtt{q})$ and $t_A^\star(\mathtt{q})$ through extensive test runs over random ensembles of instances. However, we note that verifying these assumptions experimentally is currently beyond the operational limits of the D-Wave Two device since the optimal t_A for generalized adiabaticity is expected to be smaller than the minimum programmable value Rønnow et al. (2014). Furthermore, we deemed the considerable machine time required for such a large-scale study too onerous in the context of this initial foray. Fortunately, the first limitation is expected to be lifted with the next generation of chip, at which point nothing would prevent the proper determination of a family-specific choice of $R^\star$ and $t_A^\star$. Given a specified annealing time, the number of anneals is determined by specifying r_0, the target probability of success for queries or confidence level, and measuring $r_{\mathtt{q}}$, the rate of occurrence of the ground state per repetition for the following query:

$$R^\star = \frac{\log[1 - r_0]}{\log[1 - r_{\mathtt{q}}]}. \tag{15}$$

The rate $r_{\mathtt{q}}$ depends on the family, T, and the other parameters. The minimum observed during pre-characterization should be used to guarantee the ground state is found with

at least the specified r_0. Formally, the estimated time to solution of a problem is then given by

$$\mathtt{T} = \sum_{\mathtt{q}\in\mathcal{Q}} t_A \left(\frac{\log[1-r_0]}{\log[1-r_\mathtt{q}]}\right). \tag{16}$$

The total probability of success of solving the problem in time $\mathtt{T}$ will consequently be $\prod_\mathtt{q} r_\mathtt{q}$. For the results presented in this paper, we used $R^\star = 500\,000$ and $t_A^\star = \min(t_A) = 20\ \mu$s.

We can define three different sets of qubit configurations that can be returned when the annealer is queried with $\mathtt{q}$. $\mathcal{E}$ is the set of configurations whose energy is larger than ΔE as defined in Section III of the paper. These configurations represent invalid schedules. $\mathcal{V}$ is the set of solutions with zero energy, i.e., the solutions whose makespan $\mathcal{T}$ is small enough ($\mathcal{T} \leq T - K$) that they have not been discriminated by the procedure described in the subsection on timespan discrimination. Finally, $\mathcal{S}$ is the set of valid solutions that can be discriminated ($\mathcal{T} \in (T-K, T]$). Depending on what the timespan T of the problem Hamiltonian H_T and the optimal makespan $\mathcal{T}$ are, the quantum annealer samples the following configuration space (reporting R samples per query):

$$T < \mathcal{T} \longrightarrow \quad \mathcal{V},\mathcal{S} = \emptyset \rightarrow E_0 > \Delta E, \tag{17}$$

$$\mathcal{T} \in (T-K, T] \longrightarrow \quad \mathcal{V} = \emptyset \rightarrow E_0 \in (0, \Delta E], \tag{18}$$

$$\mathcal{T} \leq T-K \longrightarrow \quad \mathcal{E},\mathcal{V},\mathcal{S} \neq \emptyset \rightarrow E_0 = 0. \tag{19}$$

Condition (18) is the desired case where the ground state of H_T with energy E_0 corresponds to a valid schedule with the optimal makespan we seek. The ground states corresponding to conditions (17) and (19) are instead, respectively, invalid schedules and valid schedules whose makespan could correspond to a global minimum (to be determined by subsequent calls). The above-described assumption ii) is essential to justify aborting the search when case (18) is encountered. If R and t_A are appropriately chosen, the ground state will be preferentially found instead of all other solutions such that one can stop annealing reasonably soon (i.e., after a number of reads on the order of $R^\star$) in the absence of the appearance of a zero-energy solution. We can then declare this minimum-energy configuration, found within $(0, \Delta E]$, to be the ground state and the associated makespan and schedule to be the optimal solution of the optimization problem. On the other hand, we note that if $K = 0$, a minimum of two calls are required to solve the problem to optimality, one to show that no valid solution is found for $T = \mathcal{T} - 1$ and one to show that a zero-energy configuration is found for $T = \mathcal{T}$. While for cases (18) and (19) the appearance of an energy less than or equal to ΔE heuristically determines the trigger that stops the annealing of H_T, for case (17), we need to have a prescription, based on pre-characterization, on how long to anneal in order to be confident that $T < \mathcal{T}$. While optimizing these times is a research program on its own that requires extensive testing, we expect that the characteristic time for achieving condition (18) when $T = \mathcal{T}$ will be on the same order of magnitude for this unknown runtime.

Search strategy

The final important component of the computational strategy consists in determining the sequence of timespans of the calls (i.e., the ensemble $\mathcal{Q}$). Here we propose to select the queries based on an optimized binary search that makes informed dichotomic decisions based on the pre-characterization of the distribution of optimal makespans of the parametrized ensembles as described in the previous sections. More specifically, the search is designed based on the assumption that the JSP instance at hand belongs to a family whose makespan distribution has a normal form with average makespan $\langle\mathcal{T}\rangle$ and variance σ^2. This fitted distribution is the same $\mathcal{P}_p$ described in Figure 2-a of the main text whose tails have been cut off at locations corresponding to an instance-dependent upper bound T_{max} and strict lower bound T_{min} (see the following section on bounds).

Once the initial T_{min} and T_{max} are set, the binary search proceeds as follows. To ensure a logarithmic scaling for the search, we need to take into account the normal distribution of makespans by attempting to bisect the range $(T_{\text{min}}, T_{\text{max}}]$ such that the probability of finding the optimal makespan on either side is roughly equal. In other words, T should be selected by solving the following equation and rounding to the nearest integer:

$$\text{erf}\left(\frac{T_{\text{max}} + \frac{1}{2} - \langle\mathcal{T}\rangle}{\sigma\sqrt{2}}\right) + \text{erf}\left(\frac{T_{\text{min}} + \frac{1}{2} - \langle\mathcal{T}\rangle}{\sigma\sqrt{2}}\right) = \tag{20}$$
$$\text{erf}\left(\frac{T + \frac{1}{2} - \langle\mathcal{T}\rangle}{\sigma\sqrt{2}}\right) + \text{erf}\left(\frac{T - \max(1,\, K) + \frac{1}{2} - \langle\mathcal{T}\rangle}{\sigma\sqrt{2}}\right),$$

where $\text{erf}(x)$ is the error function. For our current purpose, an inexpensive approximation of the error function is sufficient. In most cases this condition means initializing the search at $T = \langle\mathcal{T}\rangle$. We produce a query $\mathtt{q}_0$ for the annealing of H_T. If no schedule is found (condition (17)) we simply let $T_{\text{min}} = T$. If condition (19) is verified, on the other hand, we update T_{max} to be the makespan $\mathcal{T}$ of the valid found solution (which is equal to $T - \max(1,\, K) + 1$ in the worst case) for the determination of the next query $\mathtt{q}_1$. The third condition (18), only reachable if $K > 0$, indicates both that the search can stop and the problem has been solved to optimality. The search proceeds in this manner by updating the bounds and bisecting the new range at each step and stops either with condition (18) or when $\mathcal{T} = T_{\text{max}} = T_{\text{min}} + 1$. Figure 1-a shows an example of such a binary search in practice. The reason for using this guided search is that the average number of calls to find the optimal makespan is dramatically reduced with respect to a linear search on the range $(T_{\text{min}}, T_{\text{max}}]$. For a small variance this optimized search is equivalent to a linear search that starts near $T = \langle\mathcal{T}\rangle$. A more spread-out distribution, on the other hand, will see a clear advantage due to the logarithmic, instead of linear, scaling of the search. In Figure 1-b, we compute this average number of calls as a function of N, θ, and K for $N = M$ instances generated such that an operation's average execution time also scales with N. This last condition ensures that the variance of the makespan grows linearly with N as well, ensuring that the logarithmic behavior becomes evident for larger instances. For this calculation we

use the worst case when updating $T_{\max}$ due to condition (19) being met. We find that for the experimentally testable instances with the D-Wave Two device (see Figure 3-b of the main text), the expected number of calls to solve the problem is less than three (in the absence of pre-characterization it would be twice that), while for larger instances the size of $\mathcal{Q}$ scales logarithmically, as expected.

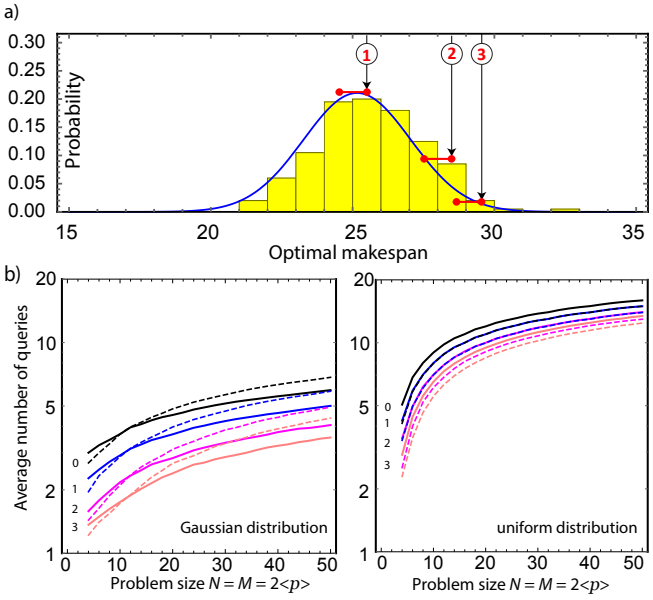

Figure 1: a) View of a guided binary search required to identify the minimum makespan over a distribution. The fitted example distribution corresponds to $N = M = 8$, fitted to a Gaussian distribution as described in the main text. We assume $K = 1$. The first attempt queries H_{26}, the second H_{29}, and the third H_{30} (the final call), following Eq. (20). b) Average number of calls to the quantum annealer required by the binary search assuming Eq. (20) (left panel) or assuming a uniform distribution of minimum makespans between trivial upper and lower bounds. Thick and dashed lines correspond to $\theta = 1$ and $\theta = 0.5$, respectively, and the numeric values associated with each color in the figure correspond to different values of K. The operations' execution times are distributed uniformly with $P_p = \{0, \ldots, N/2\}$.

JSP bounds

The described binary search assumes that a lower bound $T_{\min}$ and an upper bound $T_{\max}$ are readily available. We cover their calculation for the sake of completeness. The simplest lower bounds are the *job* bound and the *machine* bound. The job bound is calculated by finding the total execution time of each job and keeping the largest one of them, put simply

$$\max_{n\in\{1,\ldots,N\}} \sum_{i=k_{n-1}}^{k_n} p_i, \tag{21}$$

where we use the lexicographic index i for operations and where $k_0 = 1$. Similarly, we can define the machine bound as

$$\max_{m\in\{1,\ldots,M\}} \sum_{i\in I_m} p_i, \tag{22}$$

where I_m is the set of indices of all operations that need to run on machine $\mathbf{m}_m$. Since these bounds are inexpensive to calculate, we can take the larger of the two. An even better lower bound can be obtained using the iterated Carlier and Pinson (ICP) procedure described in the window shaving subsection of Section III of the main text. We mentioned that the shaving procedure can show that a timespan does not admit a solution if a window closes completely. Using shaving for different timespans and performing a binary search, we can obtain the ICP lower bound in $\mathcal{O}\big(N^2\log(N)M^2T_{\max}\log_2(T_{\max} - T_{\min})\big)$, where $T_{\min}$ and $T_{\max}$ are some trivial lower and upper bound, respectively, such as the ones described in this section. Given that the search assumes a strict bound, we need to decrease whichever bound we chose here by one.

As for the upper bound, an excellent choice is provided by another classical algorithm developed by Applegate and Cook Applegate and Cook (1991) for some finite computational effort. The straightforward alternative is given by the total work of the problem

$$\sum_{i\in\{1,\ldots,k_N\}} p_i. \tag{23}$$

The solver's limitations can also serve to establish practical bounds for the search. For a given family of problems, if instances of a specific size can only be embedded with some acceptable probability for timespans smaller than $T_{\max}^{\text{embed}}$, the search can be set with this limit, and failure to find a solution will result in $T_{\max}^{\text{embed}}$, at which point the solver will need to report a failure or switch to another classical approach.

Classical algorithms

When designing a quantum annealing solver, a survey of classical methods provides much more than a benchmark for comparison and performance. Classical algorithms can sometimes be repurposed as useful pre-processing techniques as demonstrated with variable pruning. We provide a quick review of the classical methods we use for this work as well as some details on the classical solvers to which we compare.

Variable pruning

Eliminating superfluous variables can greatly help mitigate the constraints on the number of qubits available. Several elimination rules are available and we explain below in more detail the procedure we used for our tests.

The first step in reducing the processing windows is to eliminate unneeded variables by considering the precedence constraints between the operations in a job, something we refer to as simple variable pruning. We define r_i as the sum of the execution times of all operations preceding operation O_i. Similarly, we define q_i as the sum of the execution times of all operations following O_i. The numbers r_i and q_i are referred to as the *head* and *tail* of operation O_i, respectively. An operation cannot start before its head and must leave

enough time after finishing to fit its tail, so the window of possible start times, the *processing window*, for operation O_i is $[r_i, T - p_i - q_i]$.

If we consider the one-machine subproblems induced on each machine separately, we can update the heads and tails of each operation and reduce the processing windows further. For example, recalling that I_j is the set of indices of operations that have to run on machine M_j, we suppose that $a, b \in I_j$ are such that

$$r_a + p_a + p_b + q_b > T.$$

Then O_a must be run after O_b. This means that we can update r_a with

$$r_a = \max\{r_a, r_b + p_b\}.$$

We can apply similar updates to the tails because of the symmetry between heads and tails. These updates are known in the literature as *immediate selections*.

Better updates can be performed by using *ascendant sets*, introduced by Carlier and Pinson in Carlier and Pinson (1991). A subset $X \subset I_j$ is an ascendant set of $c \in I_j$ if $c \notin I_j$ and

$$\min_{a \in X \cup \{c\}} r_a + \sum_{a \in X \cup \{c\}} p_a + \min_{a \in X} q_a > T.$$

If X is an ascendant set of c, then we can update r_c with

$$r_c = \max\left\{ r_c, \max_{X' \subset X} \left[\min_{a \in X'} r_a + \sum_{a \in X'} p_j \right] \right\}.$$

Once again, the symmetry implies that similar updates can be applied to the tails.

Carlier and Pinson in Carlier and E. (1994) provide an algorithm to perform all of the ascendant-set updates on M_j in $\mathcal{O}(N \log(N))$, where $N = |I_j|$. After these updates have been carried out on a per-machine basis, we propagate the new heads and tails using the precedence of the operation by setting

$$r_{i+1} = \max\left\{r_{i+1}, r_i + p_i\right\}, \tag{24}$$

$$q_i = \max\left\{q_i, q_{i+1} + p_{i+1}\right\}, \tag{25}$$

for every pair of operations O_i and O_{i+1} that belong to the same job.

After propagating the updates, we check again if any ascendant-set updates can be made, and repeat the cycle until no more updates are found. In our tests, we use an implementation similar to the one described in Carlier and E. (1994) to do the ascendant-set updates.

Algorithm 1 is pseudocode that describes the shaving procedure. Here, the procedure UpdateMachine(i) updates heads and tails for machine i in $\mathcal{O}(N \log(N))$, as described in Carlier and E. (1994). It returns True if any updates were made, and False otherwise. PropagateWindows() is a procedure that iterates over the tasks and checks that Eqs. (24) and (25) are satisfied, in $\mathcal{O}(NM)$.

For each repetition of the outermost loop of Algorithm 1, we know that there is an update on the windows, which means that we have removed at least one variable. Since there are at most NMT variables, the loop will run at most this many times. The internal **for** loop runs exactly M times and does work in $\mathcal{O}(N \log(N))$. Putting all of this together, the final complexity of the shaving procedure is $\mathcal{O}(N^2 M^2 T \log(N))$.

Algorithm 1 Shaving algorithm

```
1: procedure ICP_SHAVE
2:     updated ← True
3:     while updated do
4:         updated ← False
5:         for i ∈ machines do
6:             updated ← UpdateMachine(i) ∨ updated
7:         if updated then PropagateWindows()
```

Classical algorithm implementation

Brucker et al.'s branch and bound method (Brucker, Jurisch, and Sievers (1994)) remains widely used due to its state-of-the-art status on smaller JSP instances and its competitive performance on larger ones (Brinkkötter and Brucker (2001)). Furthermore, the original code is freely available through ORSEP (Brucker, Jurisch, and Sievers (1992)). No attempt was made at optimizing this code and changes were only made to properly interface with our own code and time the results.

Martin and Shmoys' time-based approach (Martin and Shmoys (1996)) is less clearly defined in the sense that no publicly available standard code could be found and because a number of variants for both the shaving and the branch and bound strategy are described in the paper. As covered in the section on shaving, we have chosen the $\mathcal{O}(n \log(n))$ variants of heads and tails adjustments, the most efficient choice available. On the other hand, we have restricted our branch and bound implementation to the simplest strategy proposed, where each node branches between scheduling the next available operation (an operation that was not yet assigned a starting time) immediately or delaying it. Although technically correct, the same schedule can sometimes appear in both branches because the search is not restricted to *active* schedules, and unwarranted idle times are sometimes considered. According to Martin and Shmoys, the search strategy can be modified to prevent such occurrences, but these changes are only summarily described and we did not attempt to implement them. Other branching schemes are also proposed which we did not consider for this work. One should be careful when surveying the literature for runtimes of a full-optimization version based on this decision-version solver. What is usually reported assumes the use of a good upper bound such as the one provided by Applegate and Cook (Applegate and Cook (1991)). The runtime to obtain such bounds must be taken into account as well. It would be interesting to benchmark this decision solver in combination with our proposed optimized search, but this benchmarking we also leave for future work.

Benchmarking of classical methods was performed on an off-the-shelf Intel Core i7-930 processor clocked at 2.8 GHz.

References

Applegate, D., and Cook, W. 1991. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing* 3(2):149–156.

Brinkkötter, W., and Brucker, P. 2001. Solving open benchmark instances for the job-shop problem by parallel head–tail adjustments. *Journal of Scheduling* 4(1):53–64.

Brucker, P.; Jurisch, B.; and Sievers, B. 1992. Job-shop (c codes). *European Journal of Operational Research* 57:132–133.

Brucker, P.; Jurisch, B.; and Sievers, B. 1994. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics* 49:107 – 127.

Carlier, J., and E., P. 1994. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research* 78(2):146 – 161.

Carlier, J., and Pinson, E. 1991. A practical use of jackson's preemptive schedule for solving the job shop problem. *Ann. Oper. Res.* 26(1-4):269–287.

Choi, V. 2008. Minor-embedding in adiabatic quantum computation: I. the parameter setting problem. *Quantum Information Processing* 7(5):193–209.

King, A. D.; Lanting, T.; and Harris, R. 2015. Performance of a quantum annealer on range-limited constraint satisfaction problems. *arXiv:1502.02098 [quant-ph]*.

Martin, P., and Shmoys, D. B. 1996. A new approach to computing optimal schedules for the job-shop scheduling problem. In Cunningham, W. H.; McCormick, S.; and Queyranne, M., eds., *Integer Programming and Combinatorial Optimization*, volume 1084 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 389–403.

Perdomo-Ortiz, A.; Fluegemann, J.; Biswas, R.; and Smelyanskiy, V. N. 2015. A Performance Estimator for Quantum Annealers: Gauge selection and Parameter Setting. *arxiv:1503.01083 [quant-ph]*.

Rieffel, E. G.; Venturelli, D.; O'Gorman, B.; Do, M. B.; Prystay, E. M.; and Smelyanskiy, V. N. 2015. A case study in programming a quantum annealer for hard operational planning problems. *Quantum Information Processing* 14(1):1–36.

Rønnow, T. F.; Wang, Z.; Job, J.; Boixo, S.; Isakov, S. V.; Wecker, D.; and Martinis, J. M. 2014. Defining and detecting quantum speedup. *Science* 345(6195):420–424.

Venturelli, D.; Mandrà, S.; Knysh, S.; O'Gorman, B.; Biswas, R.; and Smelyanskiy, V. 2015. Quantum Optimization of Fully Connected Spin Glasses. *Physical Review X* 5(3):031040.