9-2020

A hybrid framework using a QUBO solver for permutation-based combinatorial optimization

Siong Thye GOH
Singapore Management University, stgoh@smu.edu.sg

Sabrish GOPALAKRISHNAN
Singapore Management University, sabrishg@smu.edu.sg

Jianyuan BO
Singapore Management University, jybo.2020@phdcs.smu.edu.sg

Hoong Chuin LAU
Singapore Management University, hclau@smu.edu.sg

A Hybrid Framework Using a QUBO Solver For Permutation-Based Combinatorial Optimization

Siong Thye Goh[1], Sabrish Gopalakrishnan[1], Jianyuan Bo[2], and Hoong Chuin Lau[1]

[1]{*stgoh,sabrishg,,hclau*}*@smu.edu.sg*
[2]*jybo.2020@phdcs.smu.edu.sg*
UNiCEN
Singapore Management University
Singapore 679668

September 29, 2020

Abstract

In this paper, we propose a hybrid framework to solve large-scale permutation-based combinatorial problems effectively using a high-performance quadratic unconstrained binary optimization (QUBO) solver. To do so, transformations are required to change a constrained optimization model to an unconstrained model that involves parameter tuning. We propose techniques to overcome the challenges in using a QUBO solver that typically comes with limited numbers of bits. First, to smooth the energy landscape, we reduce the magnitudes of the input without compromising optimality. We propose a machine learning approach to tune the parameters for good performance effectively. To handle possible infeasibility, we introduce a polynomial-time projection algorithm. Finally, to solve large-scale problems, we introduce a divide-and-conquer approach that calls the QUBO solver repeatedly on small sub-problems. We tested our approach on provably hard Euclidean Traveling Salesman (E-TSP) instances and Flow Shop Problem (FSP). Optimality gap that is less than 10% and 11% are obtained respectively compared to the best-known approach.

1 Introduction

Combinatorial optimization problems such as the Traveling Salesman Problem (TSP), Flow Shop Problem (FSP), and Quadratic Assignment Problem (QAP) are computationally intractable, and traditionally these problems are solved by modeling them as an integer program and solved with mathematical programming solvers such as CPLEX and Gurobi. These solvers are based on the branch and bound paradigm which are exponential time algorithms.

Quantum computing offers a new approach to problem-solving. One exciting development is quantum annealing, which offers a general framework for solving combinatorial optimization problems efficiently and (provably) optimally, leveraging on quantum concepts like superposition and quantum tunneling (that enables a solution to escape from local optimality). While Quantum Annealing in its true form is still in the nascent stage of development, special quantum hardware such as D-Wave's quantum annealer has been developed that demonstrates effectiveness in solving targeted problems like Max-SAT and Max Cut (see e.g. [1]). In Quantum Annealing, a given combinatorial optimization problem is first formulated as an Ising model (or equivalently, quadratic unconstrained binary optimization problems (QUBO)) [2] and then solved using such annealing machine. Currently, all such solvers assume that the given problem is unconstrained, it has been proven in [2] that a constrained problem such as those stated in a typical mixed integer programming (MIP) can be converted to an unconstrained problem using a penalty method. Such details will be covered in the background section.

At the interface of Computer Science and Physics, we see the development of QUBO solvers such as Alpha-QUBO [3] and Fujitsu's Digital Annealer (DA) [4], which can be executed on conventional machines without relying on quantum phenomena. Unlike D-Wave that requires the problem to be embedded in the Chimera graph ([5], [6]), DA has the advantage it works with a fully connected graph; in [7], the scaling advantage of all-to-all connectivity was discussed. DA is implemented by a special (but conventional) hardware processor that enables parallel exploration of the neighborhood search space. It is based on a bit-flip simulated annealing algorithm with parallel tempering that does not require the user to specify the cooling schedule. Rather, the high and low temperatures are fixed and intermediate temperatures are adjusted with the objective of achieving an equal replica-exchange for all adjacent temperatures. It also includes a "dynamic offset" mechanism that enables the algorithm to escape from local optimality [4].

Unfortunately, all the above technologies suffer a limitation on the size of the model that can be solved directly, e.g. a limit of 2048-qubits for D-Wave, 3000 binary variables for Alpha-QUBO, and 8192 binary variables for DA. Even with prospective hardware enhancements, it is challenging to cope with large-scale combinatorial optimization problems as problem size increases. Hence, to date, it is commonly agreed that hybrid methods are needed. For instance, in [8], [9], [10], and [11], hybrid quantum-classical approaches to solving scheduling problems have been proposed.

In this paper, we focus on permutation-based combinatorial optimization problems, i.e. problems whose constraints are "sorting" constraints. For a routing problem, this corresponds to deciding the order of nodes to visit; for a scheduling problem, this corresponds to deciding the order of tasks to be served by a machine.

It is known that these problems have dedicated solvers, for example, for TSP, Concorde has been designed to specifically solve it while for FSP, there are standard heuristics such as the NEH algorithm [12]. More complex neighborhoods

for simulated annealing have also been proposed in [13]. Our goal, however, is not to compete with these solvers nor specialized algorithms. Rather, we like to illustrate that a general QUBO solver (such as DA) that is based on simulated annealing with a simple bit-flip neighborhood can be exploited to solve such a problem effectively when incorporated within an algorithm framework that we propose in this paper. By doing so, we demonstrate that a classical generic SA-based QUBO solver can be exploited to solve large-scale permutation-based optimization problems efficiently on conventional computers. While this idea on its own is not novel, if we were to extrapolate this idea, we point the way forward that someday when a quantum annealing machine is viable commercially, our proposed methodology will provide the backbone for solving large-scale combinatorial optimization problems with exponential speedup, thereby providing a strong competitor to commercial branch-and-bound-based exact solvers like CPLEX and Gurobi.

With this ambitious backdrop (which is still some distance away), our muted contributions in this paper are as follows:

1. We propose a divide-and-conquer scheme that enables us to decompose a large problem instance into instances of a manageable size so that a standard QUBO solver can be applied to these instances.

2. We propose a data scaling method to convert an instance to one with smaller cost variations while preserving the ranking of solutions for the original problem instance.

3. We propose a multilayer perceptron (MLP) approach to tune the constraint weight parameter(s) of the underlying QUBO.

4. We propose a method to project infeasible solutions obtained by the QUBO solver to feasible solutions using a polynomial-time weighted assignment algorithm.

A noteworthy point in the mind of a careful reader is that there could be many engineering choices for performing divide and conquer when applied to specific permutation-based combinatorial optimization problems. In this paper, our goal is not to prescribe a one-size-fits-all algorithm for all such problems. Rather, we illustrate that it is quite straightforward to design a decomposition approach to deal with large scale problem instances.

2 Background

A general QUBO formulation can be written as follows:

$$\min_{x \in \{0,1\}^n} x^T Q x$$

where Q_0 is an $n \times n$ matrix and x is binary decision variables.

Many combinatorial optimization problems (such as TSP, machine scheduling problems, FSP) can be formulated as quadratic binary optimization models with $m > 0$ number of linear constraints:

$$\min x^T Q_0 x$$

subject to

$$c_i^T x + d_i = 0, \forall i \in \{1, \ldots, m\},$$

which may be written as:

$$\min x^T Q_0 x$$

subject to

$$\|c_i^T x + d_i\|^2 = 0, \forall i \in \{1, \ldots, m\}$$

Applying the penalty method trick, the constraints may be converted to the following form where A_i's are the penalty terms. They introduce penalties to the objective function when the constraints to the original model are violated. It is known that when A_i's are large enough, it is theoretically equivalent to the original constrained problem:

$$\min x^T Q_0 x + \sum_i A_i \|c_i^T x + d_i\|^2$$

or more generally:

$$\min x^T Q_0 x + \sum_i A_i (x^T Q_i x + \alpha_i)$$

where Q_i is an $n \times n$ matrix.

QUBO is closely related to the Ising model in Quantum Physics. Rather than dealing with binary variables, the latter works with spins (variables) which take value from $\{-1, 1\}$ and the goal is to reach the minimum state of a Hamiltonian energy function H via a process known as quantum annealing. An Ising model can be directly and linearly transformed into a QUBO model [2].

As mentioned earlier, theoretically when A_i is sufficiently large, the model is equivalent to the original constrained problem. In practice, however, when a heuristic solver such as DA or Alpha-QUBO is applied to solve a QUBO model, its performance is sensitive to the values of A_i, as they determine the landscape of the objective function. Figure 1 illustrates that parameter tuning can improve the performance of a QUBO solver for TSP. Unfortunately, tuning is rarely discussed in the literature in the context of problem model parameters (vs algorithm-specific parameters). Typically, high-level guidance such as those presented in [2] is used as a norm. One rare example is [14] where the weight for a single parameter is tuned using a subgradient approach to solve the quadratic stable set problem.

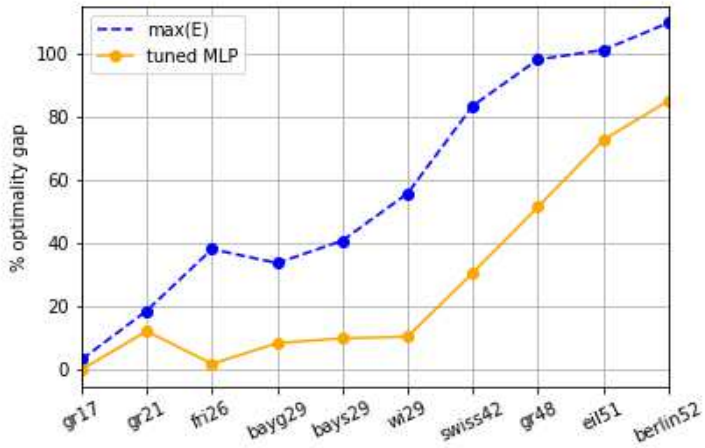

Figure 1: Effect of parameter on the traveling salesman problem.

2.1 Permutation-based Problems

A permutation-based combinatorial optimization problem involves permuting n objects to minimize a certain objective function. Common examples of permutation-based problems include the traveling salesman problem, the permutation flow shop scheduling problem, and the quadratic assignment problem. Such problems can be modeled as minimizing a quadratic objective function of the following form:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{u=1}^{n} \sum_{v=1}^{n} x_{u,i} Q_{u,i,v,j} x_{v,j} \tag{1}$$

subject to

$$\sum_{u=1}^{n} x_{u,i} = 1, \forall i \in \{1, \ldots, n\}$$

$$\sum_{i=1}^{n} x_{u,i} = 1, \forall u \in \{1, \ldots, n\}$$

where $x_{u,i}$ takes value 1 if object u is assigned to slot i and it takes value 0 otherwise. The two constraints ensure that each object is given a slot and vice versa. We call the first group of constraint the column sum constraints and the second group of constraint the row sum constraints.

We can convert this optimization formulation to a QUBO model by squaring the constraint violations and add it to the original objective function:

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{u=1}^{n}\sum_{v=1}^{n} x_{u,i}Q_{u,i,v,j}x_{v,j} \tag{2}$$

$$+ A\left[\sum_{u=1}^{n}\left(\sum_{i=1}^{n} x_{u,i} - 1\right)^2 + \sum_{i=1}^{n}\left(\sum_{u=1}^{n} x_{u,i} - 1\right)^2\right] \tag{3}$$

where A is a single parameter that we have to tune to enforce the constraint. Henceforth, we call a QUBO expressed in that form a "permutation QUBO".

Note that by considering only permutation constraints, the resulting QUBO has only one single parameter A (to be tuned).

To obtain a good performance of the QUBO solver, we need to add a few more tricks. First, we propose a data scaling procedure to scale the objective function so as to make the objective landscape smoother to improve the search effectiveness. Second, we take advantage of the permutation structure and introduce a method to ensure that the solution obtained from a QUBO solver can be easily projected to a feasible solution by solving a weighted assignment problem in polynomial time.

3 Solution Approach

Given the above motivation, we propose our algorithm framework summarized in Figure 2 to solve large scale permutation-based combinatorial problems. Our 6-stage approach is described in the following sub-sections.

3.1 Clustering

Clustering is the stage where we partition a large instance into multiple smaller instances.

This method, though generic, can be made to be more effective by taking advantage of the problem structure. For instance, if the objective is to minimize distance, we can make use of efficient procedures such as k-means clustering. Otherwise, a more general procedure such as spectral clustering [15] can be used.

Note that typically the number of clusters is a function of the number of bits so that the QUBO solver performs well in terms of solution quality and solving time. Note that the number of clusters affects the size of each sub-problem, which should not be too small as that would affect the performance when we merge the solutions of sub-problems to form a solution to the original problem. In this paper, we will use the notation k to denote the number of clusters.

Note that it is possible to treat a QUBO problem $\min x^T Qx$ as a pure optimization problem and we can fix some variables and solve for the remaining variables in each iteration. We avoided this approach as we want each sub-problem to be a special case of the overall optimization problem, allowing the human-decision maker to introduce additional human judgment to make a local improvement when necessary.

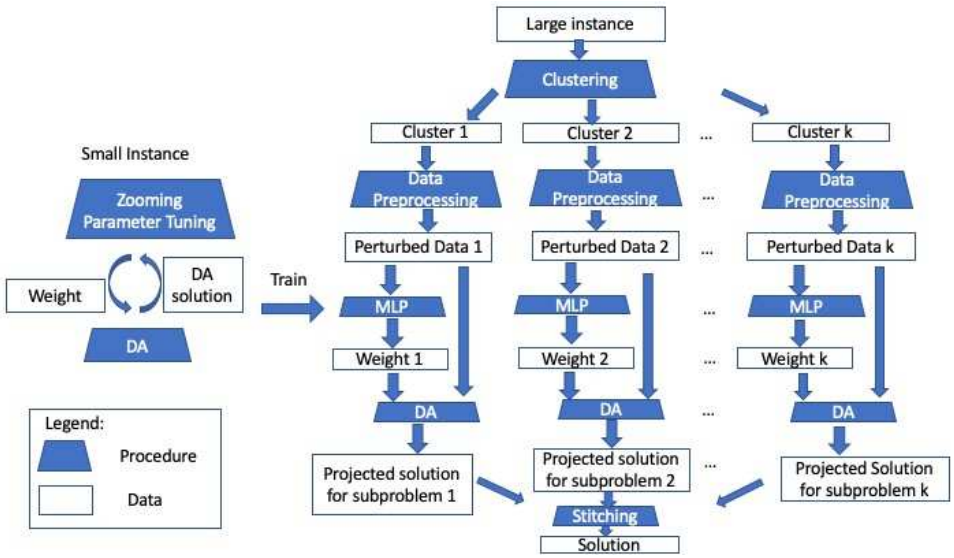

Figure 2: Proposed algorithmic framework

3.2 Data Scaling for Each Cluster

In [16], it has been shown that for the TSP problem, one can reduce all the distances to or from a particular city by a constant and preserve the ranking of the solutions. This result can be extended to the quadratic binary optimization problem with permutation constraints as follows.

Given the optimization problem (1), pick any $\hat{j} \in \{1, \ldots, n\}$ and define $\tilde{Q}$ as follows:

$$\forall u, i, v \in \{1, \ldots, n\}, \tilde{Q}_{u,i,v,j} = \begin{cases} Q_{u,i,v,j} & \text{if } j \neq \hat{j} \\ Q_{u,i,v,j} + \Delta & if j = \hat{j} \end{cases}$$

That is, this means we change those matrix entries with index $j = \hat{j}$ by a constant Δ. We call this process data scaling.

Consider the resulting scaled optimization problem:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{u=1}^{n} \sum_{v=1}^{n} x_{u,i} \tilde{Q}_{u,i,j,v} x_{j,v} \tag{4}$$

subject to $\sum_{u \in I} x_{u,i} = 1$ and $\sum_{i \in J} x_{u,i} = 1$.

The following lemma shows that the ranking of optimality is preserved under data scaling:

Lemma 1 *A solution y that is better than z for optimization problem* (1) *remains better for optimization problem* (4)*.*

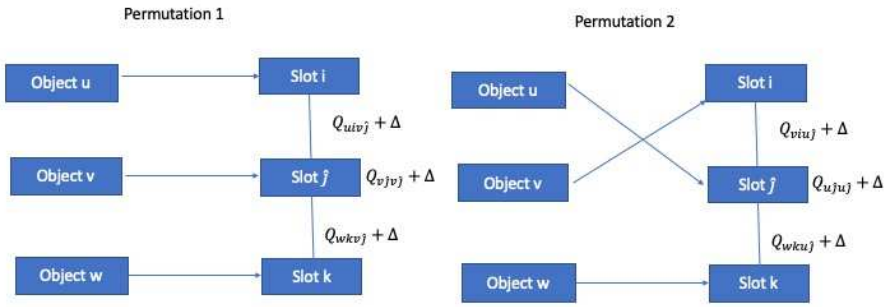

Figure 3: Simplified example of 2 permutations for the impact of data scaling.

For any pair $(u, i) \neq (v, j)$, $Q_{u,i,v,j} + Q_{v,j,u,i}$ represents the cost of assigning object u to slot i and assigning object v to slot j simultaneously. $Q_{u,j,u,j}$ is the cost of assigning object u to slot j.

Hence, by changing the entries corresponding with the index $\hat{j}$, we are in effect changing the cost of assigning object u to slot i and assigning object v to slot $\hat{j}$ simultaneously by an amount Δ. Regardless of which object is being assigned to slot $\hat{j}$, a penalty of Δ is introduced for that slot for every edge that is connected to it. Every "coupling connection" that is defined by the quadratic matrix to that slot is being penalized by Δ regardless of the permutations. The change in the final objective value of a given solution is a known multiple of Δ and hence the ranking of the permutations remains unchanged. In Figure 3, we illustrate the impact of the data scaling, notice that the net effect of a constant multiple of Δ is being added to the objective value of each permutation.

Note that the same result would hold if we change along another index/dimension as well, but for clarity, we just state the result of data scaling along one dimension. Besides changing along the slot dimension, we can also perform data scaling along the object (city/job) dimension as well.

In our work, we scale the data along all the dimensions simultaneously by minimizing the variance of the scaled matrix. We can also choose the minimal value to be non-negative if it is a desirable property.

We can generalize this further to show that data scaling works for a quadratic binary optimization problem where the sum along each dimension is equal to a constant. We formalize the proof in Appendix A. Using the notation in the Appendix, permutation problems is the special case when we set $B = C = 1$, and the index set $I = J = \{1, \ldots, n\}$.

Even though we have shown that we can change the objective value cor-

responding to certain indices by a constant and yet the two problems remain equivalent, it is interesting to determine the value to be scaled for a particular index i, i.e. Δ_i. One approach is to use the result from [17] to minimize the variance of all the scaled distances. Intuitively, this corresponds to smoothing the landscape of the objective function.

3.3 Parameter Tuning

As discussed above, it is important to tune the parameter A in Equation 3.

A naive way of running a QUBO solver to solve the original constrained problem would be to try multiple parameter values which can be extremely computationally intensive and unproductive. To search for a suitable parameter, we developed a multilayer perceptron (MLP) as a regression model to predict a suitable weight parameter value for a given instance. To our best knowledge, there is no literature that proposes machine learning to tune parameters for a QUBO formulation. A schematic diagram of the network is shown in Figure 4.

To gather sufficient training data for the MLP, we design a heuristic called the Zooming Algorithm, given as follows:

1. Partition the parameter space into bins.
2. Sample parameter from those bins and evaluate the bin's performance
3. Zoom in the bin that gives the best performance and repeat the iterations.

Each time we zoom into the bin, we increase the number of iterations. This is similar to the idea of multi-arm bandit problem where we explore the possible parameter space and then exploit the space upon finding a potentially better region that gives us better performance. The collected data are then used as the training data to build a neural network model.

Note that once the machine learning model has been built, we no longer require to try multiple parameter values; rather the trained model would output one parameter value for the QUBO model. Hence, this would reduce the computational cost significantly.

Even with machine learning, we can further improve the computational efficiency by narrowing the initial range of the parameter search space for the penalty coefficient, A. In [3], it is suggested that for converting a constrained problem to an unconstrained problem to use a QUBO solver, we can first use a ballpark estimate of the objective value and then we explore the values from 0.75 to 1.5 times of the ballpark estimate and we go through multiple rounds of iterations to improve the solutions. Empirically, for TSP, we find that 0.75 to 1.5 time the maximum scaled distance seems to give a good estimate. The choice of maximum distance is inspired by [2].

The computational complexity is hence $O(\left(\frac{n}{k}\right)^3)$, where k is the number of clusters since we need to compute the features for a given problem instance.

3.4 Projection to the Feasible Space

While ideally, the QUBO solver should return feasible (though not optimal) solutions, this need not always be the case. In this paper, we propose a projection algorithm to map an infeasible solution to a feasible solution.

Suppose $z \in \{0,1\}^{n \times n}$ is an infeasible solution returned by a QUBO solver. To restore feasibility of the original constrained problem, we solve the following optimization problem:

$$\begin{aligned} &\min \sum_{i=1}^{n} \sum_{j=1}^{n} (x_{i,j} - z_{i,j})^2 \\ &= \min \sum_{i=1}^{n} \sum_{j=1}^{n} ((1 - 2z_{i,j})x_{i,j} + z_{i,j}) \end{aligned}$$

subject to

$$\sum_{i=1}^{n} x_{i,j} = 1, \forall j \in \{1, \ldots, n\}$$

$$\sum_{j=1}^{n} x_{i,j} = 1, \forall i \in \{1, \ldots, n\}$$

Note that here z would be a given constant and hence this reduces to the standard Weighted Assignment Problem which can be solved in $O(\left(\frac{n}{k}\right)^3)$ time with the Hungarian algorithm [18].

3.5 Stitching

Having solved the sub-problems, the stitching stage combines the solutions obtained to construct a feasible solution to the original problem.

A nice property of permutation-based problems is that any permutation of $\{1, \ldots, n\}$ is a feasible solution. Hence finding good feasible solution hinges on finding an effective scheme to stitch the solutions of the clusters together (assuming that we do not need to move items between the clusters). If the number of clusters k is small, we can enumerate the configurations to find the best configuration. Otherwise, we can view the clusters themselves as items for the macro problem which we can apply our approach recursively, where the distance between the clusters would be problem-specific. For example, for TSP, we can define the distance between two clusters to be the smallest or largest distance of a city in the first cluster with a city in the second cluster.

In general, if the cost of each sub-problem is $f\left(\frac{n}{k}\right)$, we have to trade-off between the speed of solving each sub-problem and the total time taken to stitch all the sub-problems together which is a function of n and k, $g\left(n, k\right)$. The details of f and g depend on the choice of solver and the stitching procedure.

3.6 Local Improvement

Again, there are engineering decisions to be made to improve solution quality. A typical local improvement that is applicable for many permutation-based problems is one where we can swap the positions of two items. For TSP, it has been shown widely that simple local search procedures such as 2-opt can improve solution quality significantly if we perform such operations sufficiently long enough.

In the following, we apply our framework to two permutation problems: Traveling Salesman Problem and Flow Shop Scheduling Problem.

4 Application 1: Traveling Salesman Problem (TSP)

A popular formulation of TSP is the Dantzig-Fulkerson-Johnson (DFJ) formulation [19] which involves exponentially many equations to impose subtour elimination. The standard approach for solving this formulation involves using techniques such as cutting plane methods. The conversion from this MIP to the corresponding QUBO would not be scalable as that would introduce exponentially many terms in the objective function. Alternatively, the Miller-Tucker-Zemlin (MTZ) formulation [20] involves only a quadratic number of constraints. However, such formulation would still involve slack variables and hence not suitable to directly transfer to a QUBO model.

In [2], a QUBO formulation that only involves a quadratic number of terms in the number of cities is proposed. Without loss of generality, we can focus on the case where the graph is fully connected, as we can always introduce edges of infinite distances otherwise. We let d_{uv} be the distance between city u and city v. We require n^2 variables for an n-city instance. The first subscript of x represents the city and the second indicates the order that the city is going to be visited at. That is $x_{v,j}$ is the indicator variable that the city v is the j-th city to be visited. Notice that the constraint implies that this satisfies the permutation condition.

The formulation is as follows:

$$\min_x H_B(x) + AH_A(x)$$

Here

$$H_B(x) = \sum_{(u,v)\in E} d_{uv} \sum_{j=1}^{n} x_{u,j} x_{v,j+1}$$

describes the total distance travelled and

$$H_A = \sum_{v=1}^{n} \left(1 - \sum_{j=1}^{n} x_{v,j}\right)^2 + \sum_{j=1}^{n} \left(1 - \sum_{v=1}^{n} x_{v,j}\right)^2$$

describes the constraints to be a feasible cycle.

In the clustering stage, we use the constrained k-means clustering algorithm [21]. The benefit of this approach is that we have better control over the size of the clusters suitable for the QUBO solver. By fixing the number of clusters, k, the constrained k-means clustering takes in two parameters, τ, the minimum size of the cluster and μ, the maximum size of a cluster.

After clustering, we use the variance-minimizing distance scaling to change the objective to make the objective landscape smoother.

We then use our trained MLP to predict a good weight coefficient for each given instance. Note however that the MLP is trained specifically for smaller instances. For our experiments (reported below), we training our neural network with instances of 30 or less number of cities.

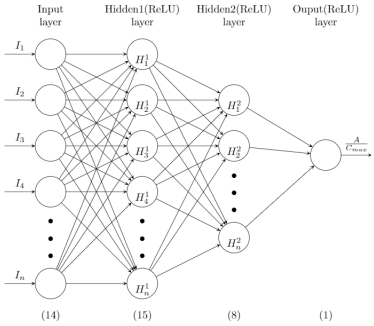

Figure 4: MLP architecture

Even though our goal is to predict optimum constraint weight for QUBO, it turns out that it is easier to predict $\frac{\text{Suitable weight}}{\text{Longest edge length}}$. We transform our data to a range between 0 and 1 and build a network comprising 362 edge weights.

The features used in the neural network are stated in Table 1. The last 3 features are well-known graph features that can be derived from eigenvalues of the distance matrix and its laplacian.

Let k be the number of clusters and largest cluster size be $|V_c|$. The stitching algorithm is outlined as follows:

1. We define the least cost flip value Δ_{ij} between all cluster pairs to be the least cost of performing 2-opt to stitch the two clusters i and j together. The time complexity is $O(k^2|E_c|^2)$.

2. To determine the ordering of stitching clusters, we solve a minimum cost Hamiltonian path problem by reusing our QUBO TSP model (except it seeks a minimum Hamiltonian path instead of cycle) with Δ_{ij} on the edges.

3. Finally, we construct the final solution for the original problem by stitching adjacent clusters derived from Step 2 via 2-opt. The stitching scheme is

Feature Name	Derivation
Iteration Number	log_{10}(DA iterations)
Minimum Edge Weight	Min. $d_{ij} \forall (i,j) \in E$
Maximum Edge Weight	Max. $d_{ij} \forall (i,j) \in E$
Mean Edge Weight	$\sum_{(i,j) \in E} \frac{d_{ij}}{\lvert E \rvert}$
Median Edge Weight	$med(d_{ij}) \forall (i,j) \in E$
Edge Weight Variance	$\sigma^2_{d_{ij} \forall (i,j) \in E}$
Number of Vertices	$\lvert V \rvert, V \in G(V,E)$
Number of Edges	$\lvert E \rvert, E \in G(V,E)$
Greedy heuristic	nearest neighbor
Two-opt heuristic	local search
Min. Spanning Tree	Kruskal's algorithm
Spectral Radius	$\rho(G)$
Condition Number	$\kappa(G)$
Algebraic Connectivity	$\Lambda(G)$

Table 1: Features and Derivation

illustrated in Figure 5.

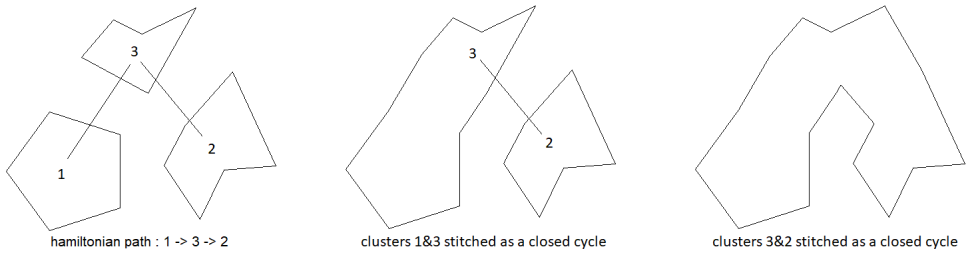

Figure 5: Stitching procedure

As the last local improvement step, we perform a standard 2-opt operation on the overall solution.

5 Application 2: Flow Shop Scheduling Problem (FSP)

Given n jobs and m machines, and each job $j, j \in \{1, 2, \ldots, n\}$ has exactly m operations where the i^{th} operation is to be performed in order by machine i. In other words, each job j is first processed by machine 1, then machine 2, and so on, until it completes its processing by machine m. The time required for

each job j on machine i, t_{ij} is given to us. The time at which job j completes its last operation shall be denoted by C_j. Each machine can only work on a single operation at any point of time. The objective is to find a non-preemptive schedule that minimizes the maximum completion time of any job by obtaining the optimal permutation of the jobs. We want to minimize $C_{\max} = \max C_j$, where C_j denotes the time at which job j completes its last operation m. Let the optimum value for a given instance be denoted as $C^*_{\max}$.

The flow shop problem is highly related to the scheduling of parts and robot movement in automated production cells nowadays. Nearly a quarter of all manufacturing, assembly, service, or information processing facilities today are set up as flow shop problems [22]. However, the problem is known to be NP-complete for more than two machines. Though many heuristic methods for solving FSP have been developed during the last decades, solving large scale FSP efficiently is still a challenge.

Permutation FSP can be formulated as a Mixed Integer Programming (MIP) problem.

$$\min C_{\max} \tag{5}$$

subject to

$$C_{jm} = S_{jm} + \sum_{i=1}^{n} x_{ij} t_{im}, j = 1, \ldots, m \tag{6}$$

$$S_{j+1,k} \geq S_{jk} + \sum_{i=1}^{n} x_{ij} t_{ik}, j = 1, \ldots, n-1; k = 1, \ldots, m \tag{7}$$

$$S_{j,k+1} \geq S_{jk} + \sum_{i=1}^{n} x_{ij} t_{ik}, j = 1, \ldots, n; k = 1, \ldots, m-1 \tag{8}$$

$$S_{i1} = 0 \tag{9}$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, \ldots, n \tag{10}$$

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, \ldots, n \tag{11}$$

The decision variables of the model, with their corresponding bound constraints are $x_{i,j} \in \{0, 1\}$, $i, j \in \{1, \ldots, n\}$ where we have

$$x_{ij} = \begin{cases} 1 & \text{, if job } i \text{ is in } j\text{-th position of the sequence} \\ 0 & \text{, Otherwise} \end{cases}$$

C_{jm} is the completion time of the j-th job on machine m. S_{jm} is the starting time of the j-th job on machine m.

Constraint [7, 8, 9] impose the rules for the starting time of each job on each machine; Constraint 6 gives the completion time of each job on the last machine as a function of its start time and its processing time. Constraint 7 says that between the starting times of consecutive jobs on a machine, there must be enough time for the first (of the two jobs) to be processed. Constraint 8 indicates that between the starting times of a job on two consecutive machines, there must be enough time for the job to be processed on the first machine. Constraint 9 says that the starting time of the first job on the first machine must be non-negative, that is, the insertion of idle time at the beginning of the schedule is allowed. Besides, constraint 10 and constraint 11 ensure that a job can only be allocated to a sequence position and that each sequence position can only be assigned to exactly one job.

From our empirical experiment, we find that the standard conversion from the MIP to QUBO directly does not work well as it involves many slack variables arising from the inequalities. An alternative QUBO formulation is needed to obtain good solutions on the QUBO solver.

Among heuristic algorithms that solve the permutation flow shop problem, the Nawaz-Enscore-Ham (NEH) heuristic [12] is known for its good performance in terms of computational complexity and solution quality. It gives the jobs that have larger total processing time higher priorities and builds the sequence by inserting the jobs in descending order of priority into the partial sequence. It preserves the partial sequence that has the minimum makespan until all the jobs have been inserted.

The analogy of the FSP with the TSP was first pointed out by Gupta [23] and has been studied by Stinson and Smith [24], Widmer and Hertz [25], and Moccellin [26]. In these papers, a distance-matrix defining a TSP-instance of n cities is set up given by the processing times of the machines. Then, the TSP instance is solved, and the solution obtained is converted to a solution of the original FSP. We use the distance matrix that is defined in [26].

The intuition behind when a distance matrix is defined is to group jobs with similar processing time close to each other to smooth the whole process and try to minimize the blocking and machine idle time. Thus, it is a natural way to try to cluster the jobs by their similarity. From the literature, we have found some suitable distances and we have listed them in Table 2. These distances enable us to convert the problem to the ATSP problem and we can use the existing QUBO formulation that we have discussed earlier even though the underlying problem that we discussed no longer has the Euclidean structure. The same data scaling, parameter tuning, and projection scheme can be applied. We discuss the clustering and the stitching procedure next.

Unlike the regular Euclidean TSP that we have discussed earlier, we no longer have the coordinate of the cities but we just have the distances and they need not be Euclidean. A different clustering approach is needed. Spectral clustering [15] techniques make use of the spectrum (eigenvalues) of the similarity matrix of the data to perform dimensionality reduction before clustering in

Reference	Distance d_{ij}
Stinson and Smith (1982) [24]	$r_{i,j,k} = t_{i,k-t_{j,k-1}}; 2 \le k \le m$ $r^*_{i,j,k} = t_{ik} - t_{j,k-1} + \min\{r_{ij,k-1,0}\}$
	$d_{ij} = \sum_{k=1}^{m} \lvert r_{ij,k} \rvert$
	$d_{ij} = \sum_{k=2}^{m} r^2_{ij,k}$
	$d_{ij} = \sum_{k=2}^{m} \lvert \min\{r^*_{ij,k}, 0\} \rvert$
	$d_{ij} = \sum_{k=2}^{m} \lvert r^*_{ij,k} \rvert$
	$d_{ij} = \sum_{k=2}^{m} (r^*_{ij,k})^2$
	$d_{ij} = \sum_{k=2}^{m} \max\{r_{ij,k}, 0\}$ $+2\lvert \min\{r_{i,j,k}, 0\} \rvert$ (Residual No Carry)
Widmer and Hertz (1989) [25] SPIRIT	$d_{ij} = t_{i,1}$ $+ \sum_{k=2}^{m} (m-k) \cdot \lvert t_{ik} - t_{j,k-1} \rvert + t_{jm}$
Moccellin (1995) [26]FSHOPH	$UBX_{ij}^{k+1} = \max\{0, UBX_{ij}^{k}$ $+(t_{jk} - t_{i,k+1})\}$ $k = 1, 2, \ldots, m-1$ $UBX_{ij}^{1} = 0$
	$d_{ij} = UBX_{ij}^{m}$

Table 2: The distance used in clustering of FSP jobs.

fewer dimensions. The similarity matrix is provided as an input and consists of a quantitative assessment of the relative similarity of each pair of points in the data set. We use spectral clustering to cluster the different jobs into clusters.

To construct the similarity matrix from the distance matrix, we use the following conversion

$$Similarity_{i,j} = \exp\left(\frac{-Distance_{ij}^2}{2\delta^2}\right)$$

δ here is the scaling factor which is chosen to be the average value of the distance matrix from the experiment results. After transformation, we could have all the elements in the similarity matrix to be in the range $[0, 1]$ and a higher value suggests that two jobs are more similar to each other.

After decomposition, the smaller scale problem will be solved using the same QUBO formulation mentioned previously. In other words, each smaller scale problem derived after the decomposition will be treated as an independent FSP. And they will be sent to DA simultaneously to solve. After the solution of each smaller problem retrieved from DA, a permutation of the sequence of each cluster will be conducted and the sequence with the minimum total makespan of the original problem is obtained by a brute-force search. Finally, for this problem, we permute the group of the ordering of the job clusters to stitch the job together and select the best cluster permutation.

In summary, the whole procedure for solving FSP using our framework is as follows:

1. Calculate the distance matrix of the original problem using our choice of distance metric.
2. Form the similarity matrix from the distance matrix.
3. Conduct the spectral clustering by setting a proper number of clusters.
4. Perform data scaling on each cluster.
5. Feed the decomposed problems simultaneously to DA.
6. Perform projection of solution
7. Combine all the clusters by choosing the best permutation of the clusters ordering which has the lowest makespan.

6 Numerical Experiments

In this section, we present experimental results that

1. investigate the effect of data scaling on the quality of the solution;
2. investigate the effect of the zooming method for parameter tuning; and
3. illustrate that our approach can solve euclidean TSP and permutation FSP efficiently.

6.1 Effect of Data Scaling

Data scaling reduces the ruggedness of the energy landscape. In Figure 6, we illustrate that data scaling improves the performance of TSP in that the solutions obtained will be strictly better than those without perturbation. Notice that the same data scaling scheme is used for FSP as well.

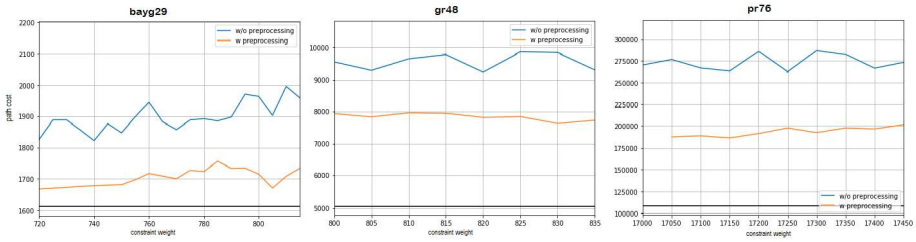

Figure 6: Improvement by performing data scaling

6.2 Effects of Parameter Tuning

In this subsection, we show the effectiveness of the zooming method to explore and exploit the parameter space. In Figure 7, we show that after a few points in the region that does not yield good performance, our method focuses on the region that provides better solutions. It overcomes the randomness of the DA solver by sampling from each bin before zooming. This parameter scheme is used in both our TSP and FSP implementations.

For $gr48$, when the weight parameter is above 800, the objective value obtained by the solver on this resulting QUBO model is 10085. In contrast, by applying our zooming algorithm, the parameter is found to be in the region around 650, which yields an objective value close to 6000, with an improvement of 40.5%. Similarly, for $bayg29$, our method gives an improvement of 21.4%, and for $berlin52$, the improvement with zooming is 42.6%. We also observe that after a region of good parameter values, as we increase the parameter value further, the performance deteriorates.

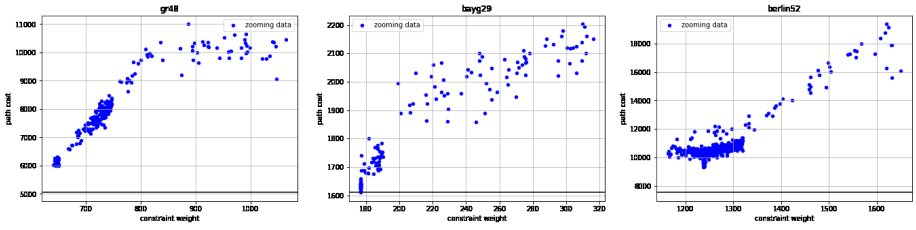

Figure 7: Parameter space search for zooming

6.3 Comparison of Our Hybrid Approaches vs Directly Using the QUBO Solver Directly

When the problem instance size is large, due to hardware limitations, we are not able to use the solver directly. Hence there is a need for a divide and conquer approach. Another advantage of doing so would be when the problem size is small, the search space for the QUBO is smaller and optimality for each sub-problem are easier to be obtained.

In Table 3, we illustrate the difference between direct application of the QUBO solver and the effect if our hybrid framework is being adopted where we solve TSP instances up to 90 cities, the limitation of our QUBO solver.

If we directly solve the problem by setting the penalty parameter to the maximum distance directly, the search space is so huge that the solving time does not scale well, we gain a saving of 82% by adopting a divide and conquer hybrid approach in terms of computational time. The quality solution is also better, typically, the hybrid approach obtains objective value that is half of the approach where we directly use the QUBO solver.

Instance	Without Hybrid Obj	Without Hybrid Comp. Time/s	Hybrid Obj	Hybrid Comp. Time/s
wi29	41538	23.3	28847	36.7
eil51	845	89.1	441	41.4
st70	1956	312.0	699	49.3
pr76	317417	305.0	124445	56.2
eil76	1511	302.4	577	53.6
Tnm52	1253662	89.5	872505	43.9
Tnm55	1629121	102.4	1118562	45.7
Tnm58	1562994	109.9	1278238	48.2
Tnm61	1607390	119.6	957370	50.9
Tnm64	1678207	153.8	1077396	52.7
Tnm67	2244710	256.8	1203115	52.5
Tnm70	2521951	272.4	1394978	54.3
Tnm73	2881902	287.9	1459234	54.9
Tnm76	3081522	306.6	1693737	57.5
Tnm79	3119744	325.1	1826576	59.0
Tnm82	3502029	341.8	1926941	62.2
Tnm85	3677904	367.6	2141045	65.6
Tnm88	4320512	391.0	2238421	69.8

Table 3: Comparison of using our hybrid framework cs the effect of directly applying the QUBO solver to solve the problem.

Instance	Avg Optimality Gap /%	Avg Runtime /s
size ≤ 100	4.3	58
$100 <$ size ≤ 318	8.1	104
$318 <$ size ≤ 1084	12.7	4165

Table 4: Comparison of results with optimal solutions from TSPLIB dataset

6.4 Results on Solving E-TSP

In all experiments reported, the DA is called in parallel tempering mode with 10^8 iterations. We use Python 3.7 as the programming language and the python HTTP library "requests" for WebAPI communications with the DA Server with the limitation of 8192 bits. The minimal cluster size τ is set to be 7 and the maximum cluster size μ is fixed at 30.

First, we evaluate our approach experimentally on data sets taken from TSPLIB. An abridged version of numerical results is shown in Table 2 (in the interest of space). We observe that for input size up to 318 cities, we achieve an average optimality gap of less than 10%. Note that the reported time is the total time taken for the entire process and not just the time to solve the QUBO models.

Next, we experimented with the computationally hard instances from [27] for which the best TSP exact solver Concorde does not scale well. In contrast, Table 4 shows that our approach remains computationally efficient compared to other existing solvers and the optimality gap is very small.

6.5 Results on Solving FSP

Here, the data sets are taken from [28]. We derive the makespans obtained by different metrics for clustering and compare them with the performance of the NEH algorithm.

By experiments, we selected the number of the clusters with the following setting:

$$\text{number of clusters} = \alpha + \lfloor \frac{\text{number of jobs}}{k} \rfloor$$

$$\alpha \in [1, 2, 3], k \in [25, 30]$$

By using these settings according to the size of the actual problem, we could derive a reasonable cluster size of around 35 which helps solve the original problem.

Size	Optimal	Concorde Time/s	Ours	Opt Gap /%	Runtime /s
$52 \leq$ size ≤ 100	NA	Avg 177.59	NA	Avg 4.73	Avg 75
103	1412229	478	1466850	3.87	68
106	1469617	761	1514937	3.08	69
109	1527709	1068	1707387	11.76	70
112	1585157	1721	1630932	2.89	69
115	1641752	1523	1662269	1.25	71
118	1698486	2261	1754417	3.29	76
121	1755689	2329	1829660	4.21	82
124	1813276	3082	1871091	3.19	82
127	1871162	3787	1913414	2.26	94
130	1928734	4332	1957868	1.51	94
133	1944317	6254	1993380	2.52	95
136	2002445	6912	2049274	2.34	98
139	2060637	9347	2107730	2.29	103
142	2118758	9980	2184781	3.12	108
145	2177169	12990	2332582	7.14	113
148	2235669	18144	2265907	1.35	122
151	2293265	18866	2338999	1.99	126
154	2350345	24972	2428242	3.31	136
157	2407153	29190	2425818	0.78	142
160	2463857	34022	2510871	1.91	148
163	2485463	56671	2513968	1.15	154
166	2543466	53752	2580772	1.47	160
169	2600546	72038	2633027	1.25	169
172	2657369	61625	2691044	1.27	180
175	2714530	80703	2809882	3.51	88
178	2771953	110158	2845337	2.65	92
181	2829701	117373	2917137	3.09	101
184	2887675	191357	2928737	1.42	110
187	2945939	249657	2999165	1.81	103
190	3004259	337307	3033477	0.97	115
193	3023551	263334	3061626	1.26	119
196	3081566	300945	3132539	1.65	128
199	3139778	411222	3208316	2.18	133

Table 5: Comparison with Concorde for Large Tnm instances. Concorde results are taken from the authors' website at `http://www.or.uni-bonn.de/~hougardy/HardTSPInstances.html`

The results of two sample problem sets are shown in Tables 6 and 7, which show that we achieve very competitive results in comparison with NEH. We summarize the optimality gap and the run time performance for instances of various sizes in Table 8 and Table 9 respectively. It is clear that when the instance size is more than 200 jobs, our decomposition scheme has an advantage in terms of computational time, where we only require 12% of the time required by NEH at the expense of at most 11% of performance deterioration compared to NEH algorithm.

Test Case	NEH	Stinson (1982) H2	Stinson (1982) H6	SPIRIT	FSHOPH
1	1160	**1128**	1176	1188	1158
2	1157	**1166**	1173	**1166**	1177
3	1149	**1143**	1146	1173	1155
4	1049	1120	**1109**	1115	1103
5	1126	**1113**	1126	1141	1139
6	1104	1099	1100	**1098**	1129
7	1117	1164	**1162**	1179	1181
8	1129	1150	**1131**	1158	1151
9	1052	**1076**	1091	1117	1091
10	1118	1158	1146	**1116**	1147
avg %	0 %	**1.46%**	1.83%	2.65%	2.46%

Table 6: Result for 10 jobs 10 machines instances. The final row indicate the performance gap with NEH as the comparison target.

We also compare our approach with optimal solutions obtained by CPLEX, and results show an optimality gap of within 11% for all the above instances.

7 Conclusion

In this paper, we proposed a divide and conquer framework aimed at exploiting a heuristic QUBO solver to solve large-scale permutation-based combinatorial optimization problems. This approach yields very small optimality gaps on TSP and FSP benchmark instances. We are currently experimenting with QAP using our framework and will release results in the near future.

On future works, we like to extend our framework to a larger class of problems whose constraints are of the form $\sum_{i \in I} x_{i,j} = B$ and $\sum_{i \in I} x_{i,j} = C$. We have shown that our data scaling method can be readily be extended, at the cost of solving a least square problem (see Annex A). For projection methods, this can be reduced to solving an MIP but it seems to be an overkill. Our proposed parameter tuning approach is rudimentary and potentially replaceable with more sophisticated hyperparameter tuning approaches in the literature.

Test Case	NEH	Stinson (1982) H2	Stinson (1982) H6	SPIRIT	FSHOPH
1	1820	1884	**1840**	1863	1912
2	1608	1683	**1665**	1703	1684
3	1730	1802	**1767**	1812	1840
4	1791	1864	1858	1879	**1847**
5	1743	1786	**1766**	1795	1850
6	1878	1909	**1877**	1927	1911
7	1749	1807	**1782**	1797	1812
8	1736	**1795**	1840	1830	1860
9	1735	1819	**1781**	1826	1794
10	1666	1739	1745	**1721**	1770
avg %	0 %	3.65%	**2.71%**	4.02%	4.76%

Table 7: Result for 30 jobs 5 machines instances. The final row indicate the performance gap with NEH as the comparison target.

Test Case	NEH/%	Stinson (1982) H2 /%	Stinson (1982) H6/%	SPIRIT/%	FSHOPH /%
(10,5)	0	-0.01	0.35	2.38	-0.37
(10,10)	0	1.46	1.83	2.65	2.46
(20, 5)	0	3.03	3.67	3.21	5.21
(20, 10)	0	5.57	4.79	6.12	6.89
(30, 5)	0	3.65	2.71	4.02	4.76
(30, 10)	0	7.50	8.30	8.56	10.22
(200, 20)	0	10.92	10.44	11.75	10.31
(200, 40)	0	10.67	9.73	10.71	9.59
(200, 60)	0	9.34	9.29	9.70	8.55

Table 8: Average makespan optimality gap for each FSP sizes. Note that a negative number indicate that we found a solution that is better than NEH algorithm.

We conclude with two reflective remarks. First, even though the experimental results we exhibit in this paper are not very significant, philosophically this paper makes progress on an increasingly popular idea of applying machine learning to combinatorial optimization. Given a new problem, algorithm design (such as the search neighborhood, cooling schedule, etc) is relegated to a good QUBO formulation. This however gives rise to the challenge of collecting a large number of instances for parameter tuning. That said, in the era of fast and effective machine learning, we believe that the burden of the combinatorial

Test Case	NEH/s	Stinson (1982) H2/s	Stinson (1982) H6/s	SPIRIT/s	FSHOPH/s	Avg /s
(10,5)	0.01	5.21	4.48	5.26	5.13	5.02
(10,10)	0.01	4.51	4.51	4.63	4.85	4.62
(20, 5)	0.05	5.09	5.17	5.05	5.00	5.08
(20, 10)	0.07	5.11	5.13	5.05	4.99	5.07
(30, 5)	0.13	7.08	7.08	7.07	6.81	7.01
(30, 10)	0.19	7.04	7.06	7.03	6.81	7.00
(200, 20)	69.27	38.50	29.10	26.13	33.06	31.70
(200, 40)	133.96	26.50	19.43	20.11	22.77	22.20
(200, 60)	197.48	27.21	21.03	21.26	26.83	24.08

Table 9: Average time(s) for each FSP sizes. The last column indicates the average run time using various distance metric.

search will increasingly fall on the machine learning task and the use of generic solvers (QUBO solvers, SAT solvers, mathematical programming solvers) rather than on clever algorithm design and laborious manual tuning.

Second, while we have experimented with the Fujitsu's Digital Annealer as the QUBO solver, our framework applies to other heuristic QUBO solvers and even quantum annealers. We hope when quantum annealers become commercially viable, this framework could be utilized for solving large-scale combinatorial optimization problems that surpass computationally what exact solvers like CPLEX and Gurobi are capable of doing today.

References

[1] S. E. Venegas-Andraca, W. Cruz-Santos, C. McGeoch, and M. Lanzagorta, "A cross-disciplinary introduction to quantum annealing-based algorithms," *Contemporary Physics*, vol. 59, no. 2, pp. 174–197, 2018.

[2] A. Lucas, "Ising formulations of many np problems," *Frontiers in Physics*, vol. 2, p. 5, 2014.

[3] F. Glover, G. Kochenberger, and Y. Du, "A tutorial on formulating and using qubo models," *arXiv preprint arXiv: 1811.11538*, 2018.

[4] M. Aramon, G. Rosenberg, E. Valiante, T. Miyazawa, H. Tamura, and H. Katzgrabeer, "Physics-inspired optimization for quadratic unconstrained problems using a digital annealer," *Frontiers in Physics*, vol. 7, p. 48, 2019.

[5] P. Date, R. Patton, C. Schuman, and T. Potok, "Efficiently embedding qubo problems on adiabatic quantum computers," *Quantum Information Processing*, vol. 18, no. 4, p. 117, 2019.

[6] A. Lucas, "Hard combinatorial problems and minor embeddings on lattice graphs," *Quantum Information Processing*, vol. 18, no. 7, p. 203, 2019.

[7] R. Hamerly, T. Inagaki, P. L. McMahon, D. Venturelli, A. Marandi, T. Onodera, E. Ng, C. Langrock, K. Inaba, T. Honjo *et al.*, "Experimental investigation of performance differences between coherent ising machines and a quantum annealer," *arXiv preprint arXiv:1805.05217*, 2018.

[8] T. T. Tran, M. Do, E. G. Rieffel, J. Frank, Z. Wang, B. O'Gorman, D. Venturelli, and J. C. Beck, "A hybrid quantum-classical approach to solving scheduling problems," in *Ninth Annual Symposium on Combinatorial Search*, 2016.

[9] T. T. Tran, Z. Wang, M. Do, E. G. Rieffel, J. Frank, B. O'Gorman, D. Venturelli, and J. C. Beck, "Explorations of quantum-classical approaches to scheduling a mars lander activity problem," in *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

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

[11] X. Liu, H. Ushijima-Mwesigwa, A. Mandal, S. Upadhyay, I. Safro, and A. Roy, "On modeling local search with special-purpose combinatorial optimization hardware," *arXiv preprint arXiv:1911.09810*, 2019.

[12] M. Nawaz, E. E. Enscore Jr, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, 1983.

[13] P. Tian, J. Ma, and D.-M. Zhang, "Application of the simulated annealing algorithm to the combinatorial optimisation problem with permutation property: An investigation of generation mechanism," *European Journal of Operational Research*, vol. 118, no. 1, pp. 81–94, 1999.

[14] S. Karimi and P. Ronagh, "A subgradient approach for constrained binary optimization via quantum adiabatic evolution," *Quantum Information Processing*, vol. 16, no. 8, p. 185, 2017.

[15] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in neural information processing systems*, 2002, pp. 849–856.

[16] M. Held and R. M. Karp, "The traveling-salesman problem and minimum spanning trees," *Operations Research*, vol. 18, no. 6, pp. 1138–1162, 1970.

[17] S. Wang, W. Rao, and Y. Hong, "A distance matrix based algorithm for solving the traveling salesman problem," *Operational Research*, pp. 1–38, 2018.

[18] R. Jonker and A. Volgenant, "A shortest augmenting path algorithm for dense and sparse linear assignment problems," *Computing*, vol. 38, no. 4, pp. 325–340, 1987.

[19] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the Operations Research Society of America*, vol. 2, no. 4, pp. 393–410, 1954.

[20] C. Miller, "Integer programming formulation of traveling salesman problems. jacm 1960; 7 (4): 326–329. desrochers m, laporte g. improvements and extensions to the miller-tucker-zemlin subtour elimination constraints," *Operations Research Letters*, vol. 10, pp. 27–36, 1991.

[21] P. S. Bradley, K. P. Bennett, and A. Demiriz, "Constrained k-means clustering," *Microsoft Research, Redmond*, vol. 20, 2000.

[22] M. Pinedo, *Scheduling*. Springer, 2012, vol. 29.

[23] J. N. Gupta, "A general algorithm for the n× m flowshop scheduling problem," *International Journal of Production Research*, vol. 7, no. 3, pp. 241–247, 1968.

[24] J. P. Stinson and A. W. Smith, "A heuristic prorammin procedure for sequencin the static flowshop," *International Journal Of Production Research*, vol. 20, no. 6, pp. 753–764, 1982.

[25] M. Widmer and A. Hertz, "A new heuristic method for the flow shop sequencing problem," *European Journal of Operational Research*, vol. 41, no. 2, pp. 186–193, 1989.

[26] J. V. Moccellin, "A new heuristic method for the permutation flow shop scheduling problem," *Journal of the Operational Research Society*, vol. 46, no. 7, pp. 883–886, 1995.

[27] S. Hougardy and X. Zhong, "Hard to solve instances of the euclidean traveling salesman problem," *arXiv preprint arXiv:1808.02859*, 2018.

[28] E. Vallada, R. Ruiz, and J. M. Framinan, "New hard benchmark for flowshop scheduling problems minimising makespan," *European Journal of Operational Research*, vol. 240, no. 3, pp. 666–677, 2015.

Appendix A Data Scaling for Optimization Problem with Constant Row Sum and Constant Column Sum

In this appendix, we go beyond the setting of permutation constraint setting and discuss the data scaling for general optimization problem with constant row sum and constant column sum.

Consider the following quadratic programming problem with binary variables.

$$\min \sum_{\substack{u,v\in I\\ i,j\in J}} x_{u,i}Q_{u,i,v,j}x_{v,j} \tag{12}$$

subject to $\sum_{u\in I} x_{u,i} = B$ and $\sum_{i\in J} x_{u,i} = C$ where $|J|B = |I|C$. That is we impose the conditions that the row sum is a constant and the column sum is another constant and the problem is feasible.

Let $\hat{j} \in J$. Consider the optimization problem with scaled data:

$$\min \sum_{\substack{u,v\in I\\ i,j\in J}} x_{u,i}\tilde{Q}_{u,i,v,j}x_{v,j} \tag{13}$$

subject to $\sum_{u\in I} x_{u,i} = B$ and $\sum_{j\in J} x_{u,i} = C$ where

$$\forall u, v \in I, \forall J \in I \tilde{Q}_{i,j,u,v} = \begin{cases} Q_{u,i,v,j} & \text{if } j \neq \hat{j} \\ Q_{u,i,v,j} + \Delta & if j = \hat{j} \end{cases}$$

The following lemma shows that the ranking of optimality is preserved under data perturbation:

Lemma 2 *A solution y that is better than z for optimization problem* (12) *remains better for optimization problem* (13).

Proof 1 *It suffices to show that the difference in the objective values of the two solutions y and z remain the same before and after data scaling. More precisely,*

$$\sum_{\substack{u,v\in I\\ i,j\in J}} z_{u,i}Q_{u,i,v,j}z_{v,j} - \sum_{\substack{u,v\in I\\ i,j\in J}} y_{u,i}Q_{u,i,v,j}y_{v,j} = \sum_{\substack{u,v\in I\\ i,j\in J}} z_{u,i}\tilde{Q}_{u,i,v,j}z_{v,j} - \sum_{\substack{u,v\in I\\ i,j\in J}} y_{u,i}\tilde{Q}_{u,i,v,j}y_{v,j}.$$

By definition of $\tilde{Q}$, we have,

$$\begin{aligned}
&\sum_{\substack{u,v\in I\\ i,j\in J}} z_{u,i}\tilde{Q}_{u,i,v,j}z_{v,j} - y_{u,i}\tilde{Q}_{u,i,j,v}y_{v,j}\\
&= \sum_{\substack{u,v\in I\\ i,j\in J}} [z_{u,i}Q_{u,i,v,j}z_{v,j} - y_{u,i}Q_{u,i,v,j}y_{v,j}]
\end{aligned}$$

$$+\Delta \sum_{u,v\in I, i\in J} [z_{u,i}z_{v,\hat{j}} - y_{u,i}y_{v,\hat{j}}] \tag{14}$$

Since both solutions are feasible, i.e. $\sum_{u\in I} y_{u,i} = \sum_{i\in I} z_{u,i} = B$ *and* $\sum_{i\in J} y_{u,i} = \sum_{i\in J} z_{u,i} = C$*, we have*

$$\sum_{\substack{u,v\in I\\ i\in J}} y_{u,i}y_{v,\hat{j}} = \sum_{v\in I} y_{v,\hat{j}} \sum_{u\in I}\sum_{i\in J} y_{i,u} = B\sum_{u\in I} C = BC|I|$$

and similarly $\sum_{\substack{u,v\in I\\ i\in J}} z_{u,i}z_{v,\hat{j}} = BC|I|$

Hence the term (14) *is equal to 0.*

Specifically when $B = C = 1$, and the index set $I = J = \{1, \ldots, n\}$, we have the special case for permutation based optimization problems.