7-2022

# A recommendation system approach to tune a QUBO solver

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

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

Matthieu PARIZY

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

## Citation

GOH, Siong Thye; BO, Jianyuan; PARIZY, Matthieu; and LAU, Hoong Chuin. A recommendation system approach to tune a QUBO solver. (2022). *NASO 2022 proceedings: Workshop on New Architectures for Search and Optimization a satellite workshop of IJCAI-ECAI 2022, Vienna, Austria, July 24*. 22-29.
**Available at:** https://ink.library.smu.edu.sg/sis_research/7756

# A Recommendation System Approach to Tune a QUBO Solver

**Siong Thye Goh** [1] , **Jianyuan Bo** [1] , **Matthieu Parizy** [2] , **Hoong Chuin Lau** [1*]
[1]School of Computing and Information Systems, Singapore Management University
[2] Fujitsu Limited

## Abstract

There are two major challenges to solving constrained optimization problems using a Quadratic Unconstrained Binary Optimization or QUBO solver (QS). First, we need to tune both the underlying problem parameters and the algorithm parameters. Second, the solution returned from a QS might not be feasible. While it is common to use automated tuners such as SMAC and Hyperopt to tune the algorithm parameters, the initial search ranges input for the auto tuner affect the performance of the QS. In this paper, we propose a framework that resembles the Algorithm Selection (AS) framework to tune algorithm parameters for an annealing-based QS. To cope with constraints, we focus on permutation-based combinatorial optimization problems, since computing the projection to the feasible space for this class of problems can be done efficiently; and for simplicity, the number of problem parameters can be reduced to one and we fix it. Methodologically, we train a recommendation system to to learn good annealing problem parameter ranges. During testing, we search for good hyperparameter values using a recommendation system approach. To illustrate our approach experimentally, we use the Fujitsu Digital Annealer as our QUBO solver and Optuna as the auto tuner to solve the Traveling Salesman Problem.

## Introduction

Many combinatorial optimization(CO) problems are known to be NP-hard. Mixed Integer Programming solvers such as CPLEX and Gurobi can solve combinatorial optimization problems. A branch and bound paradigm, which is an exact method, takes exponential time solving such problems. Recently, there is great interest to use a quantum annealing-based approach to solve these combinatorial optimization problems as stated in Lucas [2014] and Glover *et al.* [2018]. To do so, we map these optimization problems into an Ising model, or equivalently, a QUBO (Quadratic Unconstrained Optimization Problem) formulation. However, since Quantum hardware is still in a nascent stage, to test out these ideas, non-quantum solvers such as Digital Annealer by Fujitsu [2016] and Alpha QUBO are used instead. These solvers are typically heuristic in nature, and they return multiple possible solutions.

QUBOs are optimization problems of the form:

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

In Lucas [2014] and Glover *et al.* [2018], many combinatorial optimization problems are formulated as QUBOs. In Venturelli *et al.* [2016], a decision problem for the job-shop scheduling problem is proposed where it is then can be solved via a QUBO solver (QS).

While QUBOs can be used to formulate a wide variety of combinatorial optimization problems, in this paper, we focus on permutation problems (which includes Traveling Salesman Problem, Quadratic Assignment Problem, and Flowshop Scheduling Problem). We define variable $x_{ij} \in \{0, 1\}$ as an indicator variable that takes value $1$ if object $i$ is assigned to slot $j$ and it takes value $0$ otherwise. The goal is to find an optimal assignment of the objective function that is a quadratic function of these binary variables. The optimization problems that we focus on are of the form of

$$\min_{x} x^T Qx$$

subject to $\sum_{i=1}^n x_{ij} = 1$ and $\sum_{j=1}^n x_{ij} = 1$ where these constraints describe that each object is assigned a slot and each slot has an object being assigned to it respectively.

The problem stated above cannot be solved directly by a QS due to the constraints. In Lucas [2014], Goh *et al.* [2022], Huang *et al.* [2021], penalty methods have been used to convert the problem in the form of a QUBO, we rewrite the problem formulation as:

$$\min_{x} x^T Qx + A\left[\left(\sum_{i=1}^n x_{ij} - 1\right)^2 + \left(\sum_{j=1}^n x_{ij} - 1\right)^2\right]$$

where $A$ is the penalty coefficient (or the problem parameter). It can be shown that if the penalty coefficient $A$ is set to be large enough, then the unconstrained optimization problem

*Corresponding Author

is equivalent to the corresponding constrained optimization problem. However, in practice, simulated annealing-based QS does not give a high-quality solution when the penalty coefficient is too large. Lately, there have been works to tune the penalty coefficient to improve the quality of the solution, for example, Verma and Lewis [2020], Huang *et al.* [2021], Goh *et al.* [2022], Ayodele [2022], and Diez Garc´ıa *et al.* [2022]. Empirically, it is known that $A$ determines a trade-off between optimality and feasibility. Note that the penalty method theoretically can be used in general constrained optimization problems. We focus on the permutation constraints in this paper because the projection of the solution can be computed efficiently to ensure feasibility.

While most solvers have automated mechanisms in performing annealing, some QS such as Fujitsu Digital Annealer allows the user to further fine-tune the underlying annealing parameter, which allows us to further improve the quality of the solution. These annealing hyperparameters affect the search trajectories as they would influence whether a region of solutions is explored, or whether a promising region is exploited to obtain better solutions. In this paper, we focus on tuning these algorithm parameters by fixing the problem parameters (penalty coefficients) to a large value.

In summary, we consider two forms of hyperparameters:

1. **Problem parameters**: Penalty coefficients that play a role in the trade-off between the objective function and the constraints. By fixing these problem parameters, we convert the constrained optimization problem to a QUBO. For simplicity, these parameter values are identically set, and is denoted as $A$ in our paper.
2. **Annealing Parameters**: These are algorithm parameters within the QS that define the annealing behavior to solve a given QUBO instance.

There are a good number of Bayesian-based tuners that have been used in practice; some examples of Bayesian tuners (automated tuners) are Hyperopt, Optuna Akiba *et al.* [2019], SMAC Hutter *et al.* [2011]. At the time of writing, there are emerging hybrid automated tuners such as Hyperband Li *et al.* [2017] and BOHB Falkner *et al.* [2018] which are the state-of-art hyperparameter tuning approaches for various deep learning tasks. The fundamental idea of these auto tuners is that they build a surrogate model which may be based on Tree Parzen Estimator (TPE), random forest, neural network, or some other regression model that is in turn used to suggest the next set of parameter values for both exploration and exploitation purposes. Given a QS with parameters to be tuned, one direct approach is to pass the domains of the parameters to the auto tuner and allow it to automatically suggest good hyper-parameter values.

In Figure 1, we show how an automated tuner is used to tune a QUBO solver to solve a combinatorial optimization problem. Notice that the purpose of this work is not to improve the automated tuner per se, but rather we treat the initial search range as an input that needs to be tuned. More precisely, we propose a recommendation system approach to recommend suitable algorithm parameter ranges to be fed into the automated tuner for unseen QUBO instances.

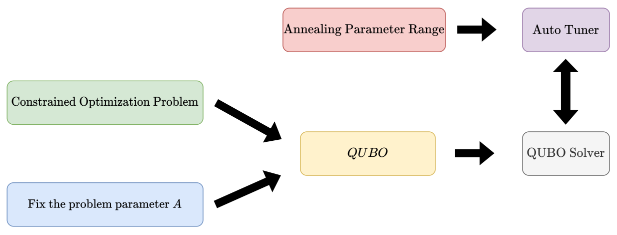
Annealing Parameter Range
Auto Tuner
Constrained Optimization Problem
QUBO
QUBO Solver
Fix the problem parameter $A$

Figure 1: A typical approach to apply a QS to solve a combinatorial optimization problem.

## Related Work

No single algorithm or a machine learning pipeline always performs better than the other algorithms. This is known as the no free lunch theorem, and it is applicable in optimization, machine learning, and statistical inference Wolpert [1996], Wolpert and Macready [1997]. However, finding the best pipeline given a new unseen instance is challenging. As categorized in Fusi *et al.* [2018], the task of leveraging on previous experiments has been explored by two different communities: The Bayesian optimization community and the algorithm selection community. In the Bayesian community, the task is commonly formulated as multi-task learning. The Multi-task Bayesian Optimization approach is used in Swersky *et al.* [2013] where a smaller data set is used to learn suitable hyperparameters of a bigger dataset. The technique of using an indicator variable to choose which dataset is used to run an experiment was presented in Schilling *et al.* [2015]. Neural networks techniques have been used in Springenberg *et al.* [2016] and Perrone *et al.* [2017].

The field of Algorithm Selection (AS) was formalized by Rice [1976] where we assign a suitable algorithm to solve suitable problem instances. Formally, $\mathcal{P}$ which denote the problem space, $\mathcal{A}$, which denote the algorithm space, and also a mapping from $\mathcal{P} \times \mathcal{A} \to \mathbb{R}$ that measure the performance if we use an algorithm on a particular problem instance.

The algorithm selection problem has been well studied for some famous problems. SATZILLA Xu *et al.* [2008], Xu *et al.* [2012] is a popular algorithm selection approach for SAT. It works by first running pre-solvers for a short amount of time and then hard instances are detected and delegated to a backup solver. Different performance models are learned for different categories of the problem instances. Stern *et al.* [2009] studies constraint solving problems and combinatorial auction winner determination problems and Malitsky and O'Sullivan [2014] projects commonly used hand-crafted meta-features used to select algorithms onto the latent space identified by their model. Mısır and Sebag [2017] further extended the work by Stern *et al.* [2009] by handling the cold start problem through nonlinear modeling of the latent factor. A recommender system proposes similar products to similar users. In the algorithm selection literature, this corresponds to recommending similar algorithms/pipelines to similar problem instances. An advantage of collaborative filtering is that during training, it is not necessary to enumerate all the configurations on all the problem instances. Imputation is frequently used in a collaborative filtering approach commonly via a latent space.

There are two approaches to collaborative filtering. Memory-based approaches depend on the similarity functions of the user and item spaces. In contrast, the model-based approach learns a lower dimension latent space of dimension $l$. A common model-based technique is based on matrix factorization approaches where the latent space is chosen to minimize the reconstruction error subject to some sparseness constraint. That is given the rating matrix $Y$. We want to find $X \in \mathbb{R}^{N \times l}, W \in \mathbb{R}^{l \times D}$ such that $Y \approx XW$. They are obtained by solving the following optimization problem:

$$\min_{X,W} \left\{ L(Y, XW) + \frac{\lambda}{2} \left( tr\left(X \cdot X^T\right) + tr\left(W \cdot W^T\right) \right) \right\}$$

where $L$ is a loss function and $\lambda$ is a regularization parameter.

Using techniques in Salakhutdinov and Mnih [2008], Lawrence and Urtasun [2009], in Fusi *et al.* [2018], a probabilistic nonlinear matrix factorization approach is used in selecting machine learning pipelines for the problem instances where we use a Gaussian Prior,

$$p(Y|X, f, \sigma^2) = \prod_{n=1}^{N} \prod_{d=1}^{D} \mathcal{N}(y_{n,d} | f_d(x_n), \sigma^2)$$

We place a Gaussian process priors over $f, p(f|X) = \mathcal{N}(f|0, K)$ where $K$ is a covariance matrix, $k(x_i, x_j) = \alpha \exp\left(-\frac{\gamma_q}{2}\|x_i - x_j\|^2\right)$ The marginal likelihood is obtained by integrating $f$ and we obtained

$$P(Y|X, \theta, \sigma^2) = \prod_{d=1}^{D} \mathcal{N}(Y_{:,d} | 0, K(X, X) + \sigma^2 I) \qquad (1)$$

where $\theta = \{\alpha, \gamma_1, \ldots, \gamma_q\}$.

A stochastic gradient approach can be used to optimize the model. That is we can solve for $\theta$ and perform imputation by minimizing the negative log-likelihood. Prediction of a new problem instance can be computed efficiently due to the prior that we have placed.

A recommender system approach has the advantage that after we perform the training stage, given an unseen instance, we can suggest suitable hyperparameter values that can be used to achieve a higher quality solution in fewer iterations. However, a recommender system framework cannot be used directly. To do so, we have to address a few challenges:

1. Convert a constrained optimization problem to an unconstrained problem by choosing a suitable penalty coefficient.
2. Obtain suitable candidates for the initial hyperparameter range.
3. Ensure feasibility of the solution returned.

Our work extends the recommender system framework for the application of a QS to solve combinatorial optimization problems. Most optimization solvers take constraints into considerations explicitly. This is not the typical setting for quantum annealing or most QS where neither the optimality nor feasibility is ensured. However, exploration of such solvers is of great importance. Similar challenges will be faced even when the quantum annealing hardware technology matures. Most quantum technologies are accessed via Cloud Services. Calling such services too frequently is expensive. By illustrating that a recommendation system is applicable for such metaheuristics, it shows that more time can be spent on exploiting good hyperparameter values.

## Proposed Framework

Using the terminology from the Algorithm Selection community, to apply a recommender system framework, we need to define the problem instance space $\mathcal{P}$, the potential algorithm space $\mathcal{A}$, and also the performance model $\mathcal{P} \times \mathcal{A} \rightarrow \mathbb{R}$:

1. Problem instance space: In our context, a problem instance is a QUBO with a defined problem parameter value.
2. Algorithm Space: In our context, the algorithm space is the proposed initial ranges of the annealing parameter values for the automated tuner.
3. Performance model: In our context, we compute the projection of the solution obtained and convert them into a relative gap w.r.t the best solution attained.

Having defined the above, the remaining steps will be to construct the CF matrix (Step 4) and perform the prediction (Step 5).

We provide the details of these components below.

### 1. Defining the problem instances

The problem instances in our context are the QUBO models derived from combinatorial optimization problems. As we have discussed earlier, $A$, the penalty coefficient determines the trade-off between optimality and feasibility. Though we acknowledge that fine-tuning problem parameter is necessary for achieving a better solution quality, we decide to not tune it but fix it with a large enough value to ensure feasibility. By doing so, we could focus on tuning the annealing parameter which has not been broadly studied. The experiments results also show that tuning annealing parameter alone could achieve good performance and even optimal solution for some of the instances.

### 2. Defining the algorithm space

Given a problem instance and an automated tuner, we describe how we can learn a range of each of the hyperparameters via a simple but effective sampling approach.

For each QUBO instance, we run the automated tuner for $m$ iterations. We then extract the hyperparameter values that correspond to the top $r\%$ of the configuration that we explored where they return feasible solutions. For each of the hyperparameters, we then compute the first and the third quartile of those best-performing hyperparameters. The pseudocode is presented in Algorithm 1.

### 3. Defining the performance measure

Optimality gap is a common metric to measure the solution quality w.r.t the optimal or the best known solution. It's not

Algorithm 1: Sampling Approach to extract potential start range for annealing hyper parameter

**Input**: QUBO solver $f$; automated tuner $T$; a QUBO instance $d$; the default initial search ranges provided by automated tuner for each of the hyperparameters $R$; the percentage of hyperparameter visited to be used, $r$; the number of times to run the automated tuner, $m$.
**Output**:Intervals of hyper parameter values that can be passed to the automated tuner.

```
1: Use the automated tuner to evaluate $T(f, d, R, m)$, that
   is we call the tuner $T$ with the solver $f$ with dataset $d$
   with the initial range being described by $R$ and record
   it. Record the hyperparameter values used and the corre-
   sponding best feasible objective function.
2: if the recorded feasible solution is non-empty then
3:     Sort the hyperparameter values according to the objec-
       tive function.
4:     Extract the hyperparameter values corresponding to
       the top $r\%$ of the recorded data.
5:     Return the first quartile and the third quartile for each
       of the hyperparameter values explored.
6: else
7:     Return the initial range without narrowing it.
8: end if
```

impractical to use the optimality gap as the performance measure, especially when conducting inference on unseen test instances where there isn't an optimal or best known solution. Since our approach tunes the parameter in an iterative manner, we could always compute the $gap$ w.r.t to the best solution achieved for each instance. This relatively measure is good enough to show the performance difference between various annealing parameters based on our experiment results.

But we are solving a constrained optimization problem and might obtain an infeasible solution from the solver. Hence, we have to process the solution obtained before we compute the gap.

To ensure feasibility, for each solution that we obtain, we compute the projection to the feasible space, a procedure that was introduced in Goh *et al.* [2022]. In terms of Hamming distance, the problem of finding the closest permutation matrix is a linear programming problem. Suppose we have obtained $y_{i,u}, i, u \in \{1, \dots, n\}$ a solution which is binary but not feasible. To obtain $x_{i,u}$ that is feasible, their Hamming distance can be written as

$$(x_{i,u}-y_{i,u})^2 = x_{i,u}^2-2x_{i,u}y_{i,u}+y_{i,u}^2 = (1-2y_{i,u})x_{i,u}+y_{i,u} \tag{2}$$

Solving the projection problem is equivalent to solving the following binary linear programming problem.

$$\min_x (1-2y_{i,u})x_{i,u} \tag{3}$$

subject to

$$\sum_{u=1}^{n} x_{i,u} = 1, \forall i \in \{1, \dots, n\} \tag{4}$$

$$\sum_{i=1}^{n} x_{i,u} = 1, \forall u \in \{1, \dots, n\} \tag{5}$$

which is just an assignment problem that can be solved in polynomial time. Once we obtain the feasible solution, we can calculate the gap accordingly.

After attaining the feasible solution, we could compute the $gap$ as the following:

$$gap = \left(\frac{\text{feasible solution} - \text{best achieved}}{\text{best achieved}}\right) \times 100\%$$

Later, we find the model trained with normalized gap is more robust and it achieves better performance. Thus, we conduct the Min-Max scaling over the $gap$ for both training and testing.

## 4. Constructing Collaborative Filtering (CF) Matrix

For each problem instance, we derive a suitable QUBO to represent it using a large enough problem parameter. After which, we find a suitable annealing parameter configuration range using Algorithm 1. With that, we can then compute the corresponding performance measure for the problem instance and annealing configuration pair. We define the Collaborative Filtering (CF) matrix or rating matrix as a matrix that records the performance of a QUBO instance given a particular annealing parameter configurations. We illustrate the CF matrix for our context of tuning hyperparameters in a QUBO solver in Figure 2.

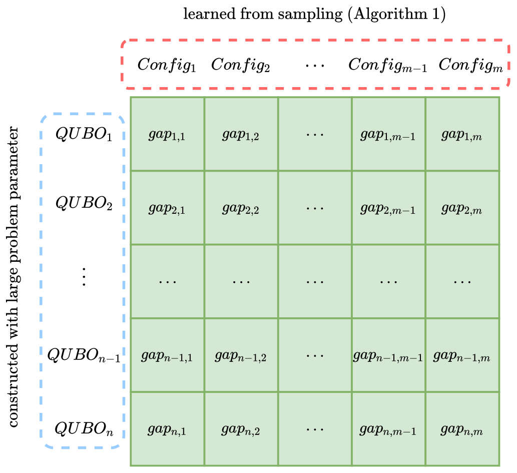
learned from sampling (Algorithm 1)
constructed with large problem parameter
$Config_1$ $Config_2$ $\cdots$ $Config_{m-1}$ $Config_m$
$QUBO_1$ $gap_{1,1}$ $gap_{1,2}$ $\cdots$ $gap_{1,m-1}$ $gap_{1,m}$
$QUBO_2$ $gap_{2,1}$ $gap_{2,2}$ $\cdots$ $gap_{2,m-1}$ $gap_{2,m}$
$\vdots$ $\cdots$ $\cdots$ $\cdots$ $\cdots$ $\cdots$
$QUBO_{n-1}$ $gap_{n-1,1}$ $gap_{n-1,2}$ $\cdots$ $gap_{n-1,m-1}$ $gap_{n-1,m}$
$QUBO_n$ $gap_{n,1}$ $gap_{n,2}$ $\cdots$ $gap_{n,m-1}$ $gap_{n,m}$

Figure 2: Example of a CF matrix where each row is a QUBO corresponding to a particular optimization problem and each column is a potential configuration.

Enumerating the performance measure for all the problem instances and the annealing configuration pair would be very expensive. To reduce the computational complexity, we use an acquisition function. An acquisition function is a function that suggests which value to explore that would likely give us the best solution. One such acquisition function is the expected improvement (EI). We also need a parameter $\tau$ that specifies the fraction of configuration to explore for each of the datasets. We describe the procedure in Algorithm 2.

```
Algorithm 2: Constructing the CF matrix
Input:Data instances of combinatorial optimization problem; the default parameter for the annealing QUBO solver; an automated tuner, $T$; initial penalty coefficient for each combinatorial optimization problem instance, $A_0$; maximum number of iteration to search for the problem parameter.
Output:A collaborative filtering matrix.
1:  for every combinatorial optimization instance, $d$ do
2:      Set the problem parameter to be $A_0(d)$, where $A_0$ can depends on the instance $d$ and construct the corresponding QUBO.
3:      if feasibility condition for the permutation constraint is satisfied then
4:          Run Algorithm 1 to obtain a suitable initial range for the instance.
5:          Compute the best feasible solution.
6:      end if
7:  end for
8:  Construct a CF matrix with the QUBO obtained as a row and the initial configurations as a column.
9:  while The CF matrix is not filled up to $\tau$ do
10:     for  each instance do
11:         Run the acquisition function to get the suggested configuration to record the best performance
12:     end for
13: end while
14: Compute the gap of the annealing configuration for each of the instances.
15: return the CF matrix.
```

Once the CF matrix has been constructed, we can use any collaborative filtering method that has a fast prediction procedure in predicting good hyperparameter values. For example, in Fusi *et al.* [2018], a probabilistic matrix factorization approach for a recommender system for selecting machine learning pipelines has been proposed. We impute the missing values by considering the mean of the gap across the instances.

## 5. Prediction Stage

Given a new problem instance, we can first use a large problem parameter to define the QUBO to represent the problem instance. Next, we have to address a common problem arising in the recommender system, the cold start problem.

The similarity is measured using the $L_1$ distance of the features. The features that we use are constructed as follows: For every QUBO instance, we compute the density of the QUBO matrix. That is, we first scale the matrix to be between $0$ and $1$, and investigate the fraction of data that are less than $0.1, 0.2, \dots, 0.9$. We also consider application-based meta-features. These meta-features used for TSP are the number of cities, the mean distance, the standard deviation of the distances, the mean of the degree, the standard deviation of the degree, and the two largest eigenvalues. After which, we use an acquisition function to predict the next value to try until we obtain $\tau\%$ of all the configurations.

```
Algorithm 3: Prediction Stage
Input:Data instances; default hyper-parameter range; the number of configurations chosen from warm start, $n$; the number of configuration to explore, $k$.
Output:Solution to the combinatorial optimization problem.
1:  Set the problem parameter to a large enough value.
2:  Construct the QUBO.
3:  Select $n$ configurations with lowest average gap from the most similar QUBO instances in the training set.
4:  Update the gap
5:  while Less than $k$ configurations have been explored do
6:      Run the acquisition function to get the suggested configuration to run the next set of configurations.
7:      Update the gap using kernel regression
8:  end while
9:  if a feasible solution is found then
10:     Return the best solution found
11: else
12:     Compute the best projected solution.
13: end if
```

We present our prediction algorithm in Algorithm 3. Note that our framework is agnostic to any specific QUBO solver or any automated tuner.

## Application

To illustrate our proposed framework, we experiment on the Traveling Salesman Problem, which is a permutation problem of finding the shortest Hamiltonian path that visits each city exactly once.

Lucas [2014] formulates TSP as follows: We let $X_{v,j}$ denote the indicator variable that city $v$ is the $j$-th city to be visited.

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

where

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

and

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

Here $H_B$ describes the length of the TSP tour and $H_A$ is an expression that would take a positive value if and only if it is infeasible. $A$, the penalty coefficient has the property that if it is increased, the solution returned by the QUBO solver tends to be feasible but not optimal. On the other hand, if it is too small, it is likely to give a better objective value but it is more likely to be not feasible for the original problem. Here, we simply fix $A = \max_{(u,v)\in E} d_{u,v}$.

## Experimental Results

In this section, we report preliminary experimental results and try to answer the following two questions:

- Could our approach produce better solutions when facing unseen test instances compared to other methods?
- Does our approach have good generalization ability for slightly out-of-distribution test instances compared to other methods?

We use the Fujitsu second-generation Digital Annealer (DA) (Matsubara *et al.* [2020]) operating in normal mode as the QUBO solver, and Optuna (Akiba *et al.* [2019]) as the automated tuner. We set the number of annealing steps to be $10^8$ and the number of replicas to be 128. The annealing hyperparameters that we tune are the temperature update interval, the decay rate, and the starting temperature. These parameters control the temperature-decreasing behavior in the annealing process. The starting temperature and the temperature update interval take a value between 0 and $10^8$. The decay rate takes a value between 0 and 1. The temperature update interval is the interval length before the temperature is updated. It takes a positive number. The update interval should be less than the number of iterations. We set the default range to be these ranges. We set the number of Optuna calls to be 60.

To train and test the recommendation system, we generated 50 random TSP instances. First, we generate a random region on the 2-dimensional space, and then sample the coordinates of all the cities using a uniform distribution and also truncated exponential distribution to generate instance size from 20 to 60 cities. We use 40 instances for training and 10 instances for testing. For test instances, we randomly select 10 of the configurations that we learned and we compute the corresponding entry.

We use random search and warm start as our baseline methods. Random search selects configurations at random. In warm start, the configurations with the smallest gap are chosen from the most similar training instances (QUBO) given the new test instance. Our goal is to show that our recommendation system could learn from the prior knowledge with training data and surpass the performance of other sampling based approaches.

The comparison is fair given the auto tuner i.e., Optuna will still be applied with identical computational resources located (same number of trials) on the configurations chosen from these methods.

### a. Comparison of the average regret

To answer the first question, we examine if our proposed recommendation system can suggest the best configurations for unseen test instances. Each instance will be given total 10 configurations and the gap is retrieved directly from the CF matrix and then we compute the average regret achieved by the recommendation system. In our paper, the regret is defined as the difference from the recommended gap from the current best configuration, hence a smaller regret is desirable. The result is shown in Table 1.

We could see that our approach has achieved the lowest average regret compared to random search and warm start.

| Methods | Ours | Random search | Warm start |
|---|---|---|---|
| Average regret | **0.05368** | 0.06201 | 0.06217 |

Table 1: Comparison of the average regret

Though the improvement is not extremely significant, the results are consistent with our recommendation system even trained with different hyperparameter settings.

We could also see that the average regret of random search is very similar to the warm start. It indicates that the manually designed QUBO features are unable to fully capture the similarity between two QUBO instances so that warm start is incapable of capturing more information than random search. In contrast, the recommendation system could potentially learn the latent representation of the QUBO instances as well as the configurations and further recommend good configurations.

### b. Comparison of the solution quality

Besides comparing the average regrets on the test instances (the results are collected by tuning each configuration with Optuna for 60 trials), we also test the performance of our recommended configuration under limited computational resources. We follow the same setting of comparing the average regret but this time we only tune each configuration with 6 Optuna trials and record the best TSP solution attained from DA.

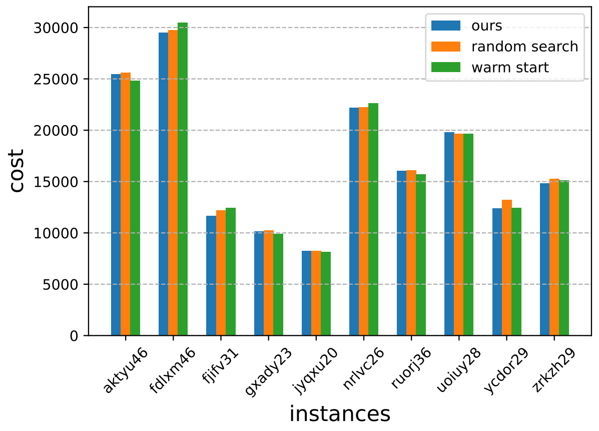

Figure 3: Solution attained using our approach, random search, and warm start on 10 test instances.

Figure 3 shows that our recommendation approach performs slightly better compared to other two baseline methods. The cost here refers to the tour length of the Hamiltonian path found. In terms of relative gap, recommended configurations has achieved average gap of $-1.692\%$ w.r.t to random search and $-0.494\%$ w.r.t warm start. It shows that the our approach can work in a more realistic experiments setting with each configuration distributed with only restricted computation budget.

We could further investigate our performance by comparing it with the best solution found in Algorithm 2 since the total number of Optuna trials for each instance are all equal to 60. Table 2 records the best solution and the number of iterations it takes to achieve the best solution.

| TSP | Ours | | Original | |
|---|---|---|---|---|
| | solution | trial | solution | trial |
| jyqxu20 | **8244** | 28 | 8288 | **11** |
| gxady23 | **10173** | **0** | 10484 | 24 |
| nrlvc26 | **22198** | 39 | 22380 | **38** |
| uoiuy28 | **19787** | 45 | 20115 | 45 |
| ycdor29 | **12378** | 22 | 12497 | **11** |
| zrkzh29 | **14820** | **12** | 14857 | 42 |
| fjifx31 | **11667** | 30 | 12705 | **18** |
| ruorj36 | **16039** | **26** | 16350 | 51 |
| aktyu46 | 25470 | **22** | **24108** | 48 |
| fdlxm46 | **29511** | **15** | 30800 | 40 |

Table 2: Best solution achieved and the number of trials it takes for 10 test instances

We observe from Table 2 that our method achieves better solutions for 9 out of total 10 test instances. Among them, 4 instances use fewer number of Optuna trials to achieve the best solution. It shows that our method could recommend good configurations that only requires fewer computational budget to achieve better solution quality.

### c. Generalization ability tested with TSPLIB instances

In the previous two sections, we show that our trained recommendation system could achieve better performance during the test on random TSP instances generated using the same distribution as the training instances. In this section we would like answer the second question to further explore the generalization ability by testing it with TSPLIB instances from Reinelt [1991]. These TSPLIB instances are commonly used benchmark which is very different from our training and testing instances.

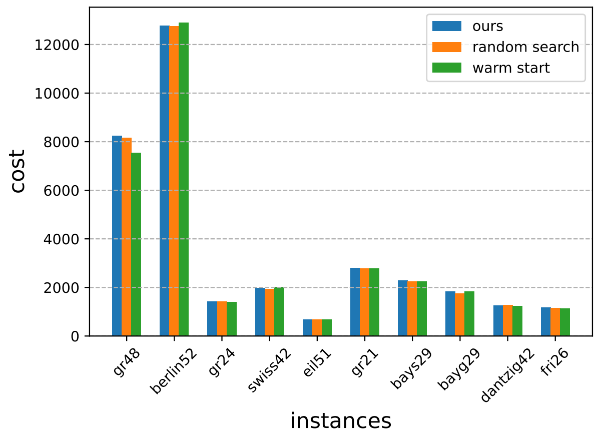

Figure 4: Solution attained using our approach, random search, and warm start on 10 TSPLIB instances.

We give each configuration 6 Optuna trials and each method could select a total of 10 configurations for testing. 10 TSPLIB instances are chosen randomly from the whole set restricting the number of cities within 20 to 60. The best solution is shown in Figure 4. Unfortunately, our method is unable to perform better than random search or a warm start. It is not surprising given the test instances are not drawn from the same distribution as our training set. In the future, we should increase the diversity of our training set to make the model generalize better for the unseen test instances.

## Discussion and Conclusion

In this paper, we propose a framework to tune the hyperparameters to attain better solutions for a QS via a recommendation system approach. From our preliminary experimental results, we learn that the recommendation system is able to suggest better configurations than competitive approaches if the training instances are drawn from a similar distribution than the testing instances. However, to date, our approach fails to tackle the issue of generality.

While we believe tuning hyperparameters can improve the performance of a QS, most of the solvers nowadays are provided in the form of a cloud service, where jobs are submitted to a queue and the solution is not returned immediately typically. Such behavior is not specific to any particular cloud service. Consequently, challenges such as jobs cancellation or long queueing time can occur. These factors make data collection an extremely manually tedious task, which severely imposes constraints on producing extensive results with massive data, unlike a typical machine learning task. We believe what is needed to produce better experimental results is a more efficient data collection scheme.

Our framework currently fixes the penalty coefficient value for each instance, and assumes that feasibility can be achieved; separately, our work involves ensuring or restoring feasibility through better tuning of these penalty values.

Other possible extensions involve minimizing the computational complexity when we make prediction in the future, and extending our approach to other combinatorial optimization problems.

## References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

Mayowa Ayodele. Penalty weights in qubo formulations: Permutation problems. In *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)*, pages 159–174. Springer, 2022.

Marcos Diez Garc´ıa, Mayowa Ayodele, and Alberto Moraglio. Exact and sequential penalty weights in quadratic unconstrained binary optimisation with a digital annealer. In *Proceedings of Genetic and Evolutionary Computation Conference Companion*. ACM, 2022.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pages 1437–1446. PMLR, 2018.

Fujitsu. Digital annealer - quantum computing technology, available today. https://www.fujitsu.com/global/services/business-services/digital-annealer/, 2016.

Nicolo Fusi, Rishit Sheth, and Melih Elibol. Probabilistic matrix factorization for automated machine learning. *Advances in neural information processing systems*, 31:3348–3357, 2018.

Fred Glover, Gary Kochenberger, and Yu Du. A tutorial on formulating and using qubo models. *arXiv preprint arXiv:1811.11538*, 2018.

Siong Thye Goh, Jianyuan Bo, Sabrish Gopalakrishnan, and Hoong Chuin Lau. Techniques to enhance a qubo solver for permutation-based combinatorial optimization. In *Genetic and Evolutionary Computation Conference Workshop on Quantum Optimization*, 2022.

Tian Huang, Siong Thye Goh, Sabrish Gopalakrishnan, Tao Luo, Qianxiao Li, and Hoong Chuin Lau. Qross: Qubo relaxation parameter optimisation via learning solver surrogates. In *IEEE International Conference on Distributed Computing Systems Workshops*, 2021.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

Neil D Lawrence and Raquel Urtasun. Non-linear matrix factorization with gaussian processes. In *Proceedings of the 26th annual international conference on machine learning*, pages 601–608, 2009.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

Andrew Lucas. Ising formulations of many np problems. *Frontiers in physics*, 2:5, 2014.

Yuri Malitsky and Barry O'Sullivan. Latent features for algorithm selection. In *Seventh Annual Symposium on Combinatorial Search*, 2014.

Satoshi Matsubara, Motomu Takatsu, Toshiyuki Miyazawa, Takayuki Shibasaki, Yasuhiro Watanabe, Kazuya Takemoto, and Hirotaka Tamura. Digital annealer for high-speed solving of combinatorial optimization problems and its applications. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 667–672. IEEE, 2020.

Mustafa Mısır and Michèle Sebag. Alors: An algorithm recommender system. *Artificial Intelligence*, 244:291–314, 2017.

Valerio Perrone, Rodolphe Jenatton, Matthias Seeger, and Cedric Archambeau. Multiple adaptive bayesian linear regression for scalable bayesian optimization with warm start. *arXiv preprint arXiv:1712.02902*, 2017.

Gerhard Reinelt. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

John R Rice. The algorithm selection problem. In *Advances in computers*, volume 15, pages 65–118. Elsevier, 1976.

Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pages 880–887, 2008.

Nicolas Schilling, Martin Wistuba, Lucas Drumond, and Lars Schmidt-Thieme. Hyperparameter optimization with factorized multilayer perceptrons. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 87–103. Springer, 2015.

Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust bayesian neural networks. *Advances in neural information processing systems*, 29:4134–4142, 2016.

David H Stern, Ralf Herbrich, and Thore Graepel. Matchbox: large scale online bayesian recommendations. In *Proceedings of the 18th international conference on World wide web*, pages 111–120, 2009.

Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Multi-task bayesian optimization. 2013.

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

Amit Verma and Mark Lewis. Penalty and partitioning techniques to improve performance of qubo solvers. *Discrete Optimization*, page 100594, 2020.

David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research*, 32:565–606, 2008.

Lin Xu, Frank Hutter, Jonathan Shen, Holger H Hoos, and Kevin Leyton-Brown. Satzilla2012: Improved algorithm selection based on cost-sensitive classification models. *Proceedings of SAT Challenge*, 2012, 2012.