Applications of quantum annealing in combinatorial optimization

Yarkoni, S.

CHAPTER 4

Real-world combinatorial optimization

There are many differences between real-world optimization problems and canonical problems in NP. While the former may build on the latter, in many cases real-world problems convolve multiple problem classes, or add additional constraints and terms required to take into account real-world scenarios. For example, the Vehicle Routing Problem is a well-known NP-hard optimization problem, which in its canonical presentation requires only a given set of locations and a number of agents to describe completely. However, in real-world applications, road networks, scheduling time windows, and availability times all must be taken into account to correctly represent the problem in practice. These real-world conditions may impose additional constraints on the search space, yet often result in equally complex problems. For instance, the Capacitated Vehicle Routing problem with Time Windows, a more realistic representation of the problem where vehicle capacities and delivery time windows are included, is still NP-hard. Thus, to address solving such combinatorial optimization problems with QUBO/Ising formulations and quantum annealing, we must introduce methods to incorporate such real-world conditions in our modeling.

In this chapter we investigate two such techniques of solving real-world problems with quantum annealing methods, each of which are fundamentally different. The first involves deriving a QUBO from a real-life optimization problem in logistics (the less-than-truckload problem). We introduce generic methods to model the various constraints and optimization terms in QUBO form to accurately capture the real-world nature of the problem. The specific limitations imposed by using quantum annealing as a method to solve these QUBOs is addressed, and techniques are proposed, implemented, and tested in this context. The (sometimes unfavorable)

scaling requirements of the problem description as a limiting factor is discussed in detail.

The second method takes an opposite approach to solving real-world optimization problems. Instead of deriving a QUBO to model the optimization problem directly, we utilize a canonical NP-complete problem– the set cover problem, which has a well-known QUBO form– as an oracle for a semi-supervised classification algorithm. We consider a variety of open source datasets of different types, and utilize various techniques to transform the data so that the task of data reconstruction (from a reference database) can be performed by the set cover QUBO. We show how the QUBO size and connectivity used to perform this task scale, and how these affect the performance of our method. Finally, we show how to use this reconstruction method to classify our data, and compare this to similar traditional classification algorithms.

4.1 Combinatorial optimization with real-world constraints

4.1.1 The shipment rerouting problem (SRP)

In order to introduce the idea of real-world applications of QUBO/Ising problems and quantum annealing methods, we derive a QUBO formulation of a real-world optimization problem as a case-study. We focus on a well-known problem in logistics: the less-than-truckload network service design. The term less-than-truckload (LTL) denotes shipments not exceeding a maximum weight significantly below a full truck load. This application is a common problem in the logistics industry, where complex delivery networks must be serviced regularly; for example, a country-wide mail delivery service is a kind of problem in this class. The transport of a single shipment in these scenarios can be described as a three step process: (a) a collecting truck run (where shipments are picked up from service locations), followed by (b) one or several linehaul truck runs (where shipments are passed through a network of connecting distribution locations) and ending with (c) a distributing truck run (a delivery to the shipments' final destination locations from a service hub). The work presented here focuses on the design of the linehaul network, or step (b). The linehaul network for LTL is made up by

the set of terminals and timetable based truck runs, connecting the terminals and thereby producing the long-hauls of all the shipments entering the network. Taking limitations on transport times into account, the forwarding of the shipments is meant to be as cost efficient as possible, measured here as the total distance each shipment must travel from origin to destination. One key factor for cost efficiency is the consolidation of multiple shipments in jointly utilized trucks, at least regarding parts of their individual linehaul paths through the network. Meaning, it may be favorable to merge multiple partially-filled trucks which service similar routes, to reduce the total distance being traveled in the network. This saving measure is represented by an increase in truck utilization. However, the consolidation of multiple shipments with different origins and destinations in jointly utilized trucks requires detours of shipments, thereby possibly increasing the distance traveled for any *individual* shipment in the network. As detours come at a cost, the network design problem searches for an optimal trade-off between detour costs and the benefit of increased truck utilization. We focus on this central trade-off decision and call this problem (the middle step in the process) the *shipment rerouting problem* (SRP). We provide an illustrative example with two shipments in Fig. 4.1.

The input to our description of the SRP includes a set of terminals, their distances between each other, and a maximum number of available trucks between any two terminals. We have a set of shipments, each with a set of possible routes of intermediate terminals throughout the network. These possible routes already comply with constraints like maximum transport time or maximum detour factor, and we consider the (weighted) graph network they represent as a fixed input to the problem. These candidate routes include the direct route from the origin to the destination of the shipment, which are also used as the default for all shipments. Other candidate routes for rerouting are constructed in a pre-processing step based on the graphical structure of the terminals and the distances between them. Thus, a subset of shipments may be rerouted through alternate routes in order to reduce the overall distance all trucks travel to deliver the shipments. Each shipment has a size metric (volume, weight, etc.), and each truck has a corresponding capacity, i.e. an upper bound for the total shipment size that can be loaded. For our purposes, we denote the shipment sizes and truck capacities with respect to volume, and refer to them as such throughout the rest of this chapter. We note that the mathematical formulations we use to derive the final QUBO equally admit other quantities.

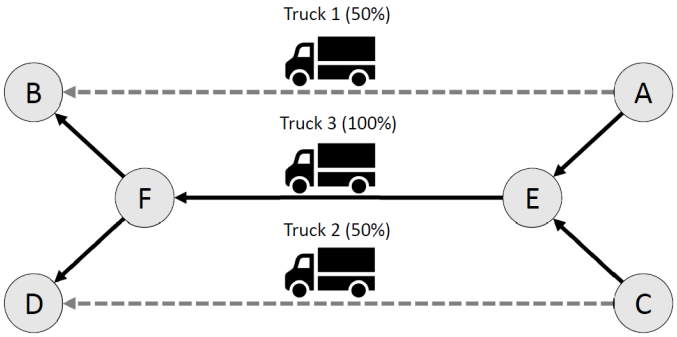

Figure 4.1: An example of the SRP with two shipments. The default routing (Trucks 1 and 2 carrying their respective shipments at 50% capacity each) is optimized by replacing Trucks 1 and 2 with a single truck (Truck 3) which can be fully utilized. The cost of rerouting each shipment to the route serviced by Truck 3 is offset by the removal of Trucks 1 and 2, thereby reducing the overall distance travelled to deliver the shipments.

In our case, a single shipment cannot be split across different routes. However, for transporting a shipment between two terminals, we may split it to distribute it on multiple trucks along the same route (this is necessary especially for shipments with large volume). Given the input to the problem, the optimization task is to decide on a route for each of the given shipments, which may include partial (or entire) overlaps between shipments. Consequently, the result includes the number of required trucks in the network, and which terminals are connected by truck runs in which frequency. Conceptually, this is similar to [98], where it was shown how to form a QUBO representation of a simple traffic flow combinatorial optimization problem. In that work, individual vehicles are given multiple candidate routes whose intersection needs to be minimized. This route-generation procedure is used here as well, but with opposite intent: our objective is to consolidate as many routes as possible. The SRP application we consider here is different from other QUBO formulations found in literature as not only the selection of routes for each shipment is variable, but also the number of trucks used on each edge along the

path is selected by the optimization. Typically, such parameters are inputs to the QUBO construction which are then used to construct the appropriate QUBO. Our formulation thus incorporates elements from both scheduling (route selection in [98]) and packing problems (canonical problems in NP [77]).

4.1.2 Constructing a MIP for the SRP

Representing the SRP as a mixed-integer program (MIP) is straightforward, as we can use multiple kinds of variables (binary, integer, real) and constrain both the search space and solutions explicitly. Therefore, we start with a MIP representation which we then later will transform to a QUBO.

First, we represent the connectivity of the terminals as a weighted directed graph G where the vertices V are the terminals and the edges E between them represent the ability to transport shipments from any single terminal to another; in other words, we have an edge $e \in E$ from a terminal a to a terminal b if there are trucks available to drive from a to b. These trucks are called the *trucks on* e and the maximum available number of trucks is denoted by $t_{\max}(e)$. The weight of e is the distance from a to b and is denoted by $d(e)$. For each shipment s, $v(s)$ denotes its volume and $R(s)$ denotes the set of all routes that can be used to transport s (*candidate routes* of s). For each edge e, $R(e)$ denotes the set of all candidate routes which pass through e. A shipment s is *scheduled* on some edge e if s is transported using an associated candidate route r containing e.

In our scenarios, we assume all trucks have the same volume capacity, which we denote by c_{vol}. Moreover, all shipments have different origin-destination pairs so that no two different shipments have common candidate routes (however, their candidate routes may overlap). Therefore, for each candidate route r, we have a unique shipment $s(r)$ that can be transported using r. These choices simplify our scenario, but result in no less of a general mathematical representation of the problem.

Our objective is to transport each shipment entered in the network along an associated candidate route such that the total distance of *all* used trucks in the network is minimized. To represent this problem by a MIP, we introduce a binary decision variable y_r for each candidate route r that is 1 if r is used to transport $s(r)$, and 0 otherwise. For each edge e, we introduce a non-negative integer variable t_e

with maximal value $t_{\max}(e)$ representing the number of used trucks on e. Thus, we represent the problem by the following MIP:

Objective: Minimize the total truck distance

$$\sum_{e \in E} d(e) \cdot t_e \tag{4.1}$$

with respect to the following constraints:

Route-shipment constraints: For each shipment s, exactly one associated candidate route is used:

$$\sum_{r \in R(s)} y_r = 1. \tag{4.2}$$

Capacity constraints: For each edge e, the total volume of all shipments scheduled on e does not exceed the total volume capacity of the used trucks on e:

$$\sum_{r \in R(e)} v(s(r)) \cdot y_r \leq c_{\text{vol}} \cdot t_e. \tag{4.3}$$

The capacity constraints ensure that on each edge e, enough trucks are used to transport all shipments scheduled on e because we can split shipments to optimally exploit the truck capacities. Note that in an optimal solution, each truck number t_e is as small as possible, namely $\left\lceil \sum_{r \in R(e)} v(s(r)) \cdot y_r / c_{\text{vol}} \right\rceil$. In that case, for each edge e, we can completely fill all used trucks on e except possibly one truck that is partially filled.

4.1.3 From MIP to QUBO

As discussed thoroughly in Sec. 2.4, contrary to a MIP, a QUBO contains only binary variables and an objective function to be minimized without explicit constraints. Thus, additional variables and constraints must be modeled with penalty factors. Here, we show how to derive such factors to emulate the MIP constraints.

Our QUBO formulation also uses the binary variables y_r for the candidate routes r. In replacement of the integer variables t_e for the edges e, we use modified binary representations of their values in the QUBO based on a concept in [77]: for each edge e, we define $T(e)$ to be the set of all powers of two less than or equal $t_{\max}(e)$, and for each $n \in T(e)$, we introduce a binary variable $t_{e,n}$ to represent the number

of used trucks on e by $\sum_{n\in T(e)} n \cdot t_{e,n}$. In this way, we can represent at least each number up to $t_{\max}(e)$, i.e. each allowed truck number. However, the highest number that can represented is $2n_{\max} - 1$ where $n_{\max}$ is the maximal value in $T(e)$. Therefore, to avoid representations of numbers greater than $t_{\max}(e)$, we reduce the coefficient $n_{\max}$ in $\sum_{n\in T(e)} n \cdot t_{e,n}$ by the surplus $s := 2n_{\max} - 1 - t_{\max}(e)$. The new expression is denoted by

$$\sum_{n\in T(e)} \overline{n} \cdot t_{e,n}, \tag{4.4}$$

i.e. we have $\overline{n_{\max}} = n_{\max} - s = 1 + t_{\max}(e) - n_{\max}$ and $\overline{n} = n$ for each $n \neq n_{\max}$. Now we can still represent each number up to $t_{\max}(e)$ but no other numbers. In our QUBO, we reformulate the total truck distance (4.1) as

$$\sum_{e\in E} d(e) \cdot \sum_{n\in T(e)} \overline{n} \cdot t_{e,n}. \tag{4.5}$$

To encode the route-shipment constraints (4.2), since that they are linear equalities, they can be added directly as $\lambda \cdot (A - B)^2$ where λ is a large penalty factor ensuring that the constraint is fulfilled at least in all optimal solutions of our QUBO. The capacity constraints (4.3), however, cannot be implemented in the same way (after reformulation using the representations (4.4)) because they are inequalities of the form $A \leq B$. However, as discussed in Sec. 2.4.1, such a constraint can be transformed into an equality $A + \ell = B$ by using a slack variable ℓ. In the context of the SRP, this slack of the capacity constraint for each edge e represents the wasted volume in the used trucks on e (*volume capacity slack* on e). However, determining the number of slack variables required must be determined from the other coefficients in the inequality. Here we consider the "packing" constraint in the QUBO formulation for the knapsack problem with integer weights in [77]:

$$H = A\left(\sum_{n=1}^{W} n y_n - \sum_{\alpha} w_\alpha x_\alpha\right)^2. \tag{4.6}$$

Here, w_α is the weight of item α, with associated decision variable x_α. The y_n variables function as unconstrained slack variables, allowing the total weight of the knapsack to be any integer less than or equal to W, as required by definition. Notice, however, that because integers are used for both the item weights and the slack variables, then all partial sums of item weights are represented by the slack

variables. In the SRP problem, the volume $v(s(r))$ is not restricted to integers, and therefore needs more consideration. A naive attempt to use the standard inequality constraint would require us to consider all partial sums of shipments through every edge e in the graph network:

$$\left(\sum_{n=1}^{|R(e)|} \left(\sum_{m=0}^{v(s(r))C_n} y_{n,m} \right) - c_{\text{vol}} \sum_{n \in T(e)} \bar{n} \cdot t_{e,n} \right)^2 = 0. \tag{4.7}$$

Here, $y_{n,m}$ would be a slack variable representing m possible sub-summations of n shipments (with ${}_{v(s(r))}C_n$ being the binomial coefficient). Clearly, this is not tractable and would result in significantly more QUBO variables than intended[1]. To overcome this problem, we discretize the shipment volumes into *bins*: We divide the capacity of each truck into the same number c_{bin} of equally sized bins. We call this c_{bin} the *bin capacity* of the trucks. Each bin can only be used for transporting one shipment and has the volume capacity $c_{\text{vol}}/c_{\text{bin}}$. Hence, for each shipment s, the number $b(s)$ of bins needed to transport s is given by $b(s) = \lceil v(s) \cdot c_{\text{bin}}/c_{\text{vol}} \rceil$. Instead of the volume capacity slacks, we now have to represent the *bin capacity slack* on each edge e, i.e. the number of unused bins in the used trucks on e. These slacks are more tractable because they are integers that can be assumed to be less than c_{bin}.

On the other hand, we must consider the case where c_{bin} is too small, and the bin volume capacity $c_{\text{vol}}/c_{\text{bin}}$ is large so that we may obtain several partially filled bins in the trucks, especially if shipments exist that are smaller than the bin volume capacity. Hence, we may not optimally exploit the truck capacities any more which may increase the number of used trucks. We can improve the situation by subdividing each bin into the same number of smaller bins.[2] Therefore, c_{bin} is a crucial parameter for the QUBO construction: more bins may lead to a better exploitation of the truck capacities, but at the cost of larger bin capacity slacks to be represented. In our experiments, we used the bin capacity 10, which was an empirically-determined compromise.

[1] Even considering duplicates in the partial sums, this is still factorially many terms in the worst case for every edge in the graph.

[2] Simply increasing the bin capacity may worsen the situation. For instance, suppose that $v(s) = c_{\text{vol}}/2$ for each shipment s so that $b(s) = \lceil c_{\text{bin}}/2 \rceil$. If $c_{\text{bin}} = 2$, then $b(s) = 1$ so that we can put two shipments into a truck. But if $c_{\text{bin}} = 3$, then $b(s) = 2$ so that we can put only one shipment into a truck.

For each edge e, we introduce a non-negative integer variable ℓ_e representing the bin capacity slack on e. Each bin capacity slack ℓ_e is less than c_{bin} so that we can represent these values in the QUBO by using a binary encoding scheme, introduced in Sec. 2.4.3. We define L to be the set of all powers of two less than c_{bin}, and for each edge e and for each $m \in L$, we introduce a binary variable $\ell_{e,m}$ such that the bin capacity slack of e is

$$\sum_{m \in L} m \cdot \ell_{e,m}. \tag{4.8}$$

In this way, we can represent at least each number less than c_{bin}, i.e. each relevant bin capacity slack. Finally, we can reformulate the capacity constraints as follows:

Capacity constraints: For each edge e, we have

$$\sum_{r \in R(e)} b(s(r)) \cdot y_r + \sum_{m \in L} m \cdot \ell_{e,m} = c_{\text{bin}} \cdot \sum_{n \in T(e)} \overline{n} \cdot t_{e,n}. \tag{4.9}$$

Similar to the route-shipment constraints, these capacity constraints are implemented in the standard way, by introducing them as a large penalty term of the form $\lambda \cdot (A - B)^2$. Putting all components together, we obtain the following formulation of the QUBO:

$$\begin{aligned} \text{Obj} = &\sum_{e \in E} d(e) \cdot \sum_{n \in T(e)} \overline{n} \cdot t_{e,n} + \lambda \cdot \sum_{s \in S} \left(\sum_{r \in R(s)} y_r - 1 \right)^2 \\ &+ \lambda \cdot \sum_{e \in E} \left(\sum_{r \in R(e)} b(s(r)) \cdot y_r + \sum_{m \in L} m \cdot \ell_{e,m} - c_{\text{bin}} \cdot \sum_{n \in T(e)} \overline{n} \cdot t_{e,n} \right)^2. \end{aligned} \tag{4.10}$$

Here, all variables are as before, and S is the set of all shipments in the problem. We must now choose a penalty factor λ to ensure that only feasible solutions are present in the global optimum of the QUBO objective, so that it is never energetically favorable to violate one of the constraints in favor of minimizing the total truck distance. In general, we may choose any λ greater than the total truck distance $d(\mathit{feas})$ of any known feasible solution *feas* (for instance, the solution transporting each shipment on its direct route). To see the correctness of this choice, consider an optimal solution *opt* and suppose that *opt* violates a constraint. Then the *opt*-value of the QUBO objective is at least λ and thus greater than $d(\mathit{feas})$.

But since *feas* is feasible, $d(feas)$ is also the *feas*-value of the QUBO objective, contradicting the optimality of *opt*.

The resulting QUBO representation now contains all the necessary terms to correctly model the SRP problem. However, it requires many more variables than the MIP in Section 4.1.2. This makes the problem more difficult to solve, both in sense of increasing the search space (in terms of number of variables) as well as the possibility of obfuscating optima via the choice in discretization (meaning, we may accidentally cause over/under filling in the trucks due to discretization). For each truck number variable t_e in the MIP, we have $|T(e)| = \lceil \log_2(t_{\max}(e) + 1) \rceil$ variables $t_{e,n}$. Additionally, we have $|L| \cdot |E| = \lceil \log_2 c_{\text{bin}} \rceil \cdot |E|$ variables $\ell_{e,m}$ to represent the bin capacity slacks. Thus, we can already see the overhead associated with conversion of a real-world optimization problem to QUBO form.

4.2 Solving real-world QUBO models

4.2.1 Generating SRP QUBOs from data

The inputs used in this work were generated from a real-world network of delivery hubs in Europe. The specific locations and distances between hubs were abstracted to comply with data protection laws, but are nonetheless representative of the original real-world network. Connections between hubs correspond to serviced routes between hubs. We used one graphical model to represent the entire hub network, and generated multiple inputs based on different numbers of shipments: 30, 50, 80, and 100 shipments. In all inputs, every shipment s_{ij} travels from one hub (v_i) to another (v_j). The direct route, $v_i \to v_j$ along e_{ij}, is always the first candidate route for s_{ij}. The other candidate routes are generated by a staggered *k-shortest path* approach: shipments are categorized by their origin-destination distance, and for each category the k shortest paths are calculated where k increases with respect to the origin-destination distance of the category. For example, shipments up to 200 km have one alternative route while shipments over 1000 km have up to 10 routes. The volume of the shipments is randomly generated using an adapted exponential distribution, resulting in many smaller shipments and few larger shipments. We show in Table 4.1 the number of QUBO

variables and the number of total terms in the QUBO for each of the problems we generated.

Shipments	Routes	QUBO variables	QUBO terms
30	223	787	4856
50	428	1526	16315
80	752	2305	40594
100	925	3318	59014

Table 4.1: Number of QUBO variables and terms needed to describe the SRP instances.

4.2.2 Comparing QUBO and MIP solvers for the SRP

To understand the impact of our choices in modeling the SRP as a QUBO, we employed the use of multiple other solvers. Here we provide a brief introduction and motivation for each solver. It is important to note that in our inputs, all shipments have different origin-destination pairs. Therefore, two different shipments cannot have common candidate routes. However, candidate routes of different shipments may overlap in some edges.

Direct shipments. We consider the "direct shipment" solution to the SRP as a simple baseline for the other solvers to beat. The direct solution is computed by routing every shipment (s_{ij}) along its most direct path (e_{ij}). Since every shipment origin/destination is unique in our instances, this equates to using one truck per edge for every shipment.

Simulated thermal annealing. We use the same simulated thermal annealing algorithm used for experiments in Ch. 2 for the MIS instances. The specific implementation of simulated annealing in this analysis was from the D-Wave Python package here [99].

Tabu search. This algorithm is another metaheuristic for combinatorial optimization, operating on the principle that searching already-discovered solutions should be actively discouraged (a "tabu list"). Individual variables' states are flipped based on their likelihood of importance in the global optimum [100]. Solutions which worsen the objective function value may be explored by the search if no other variable flip is possible, which allows for both global and local refinement of solutions. The Python package used for Tabu can be found here [99].

Gurobi. Optimal solutions and optimality bounds were produced by solving the MIP in Section 4.1.2 using Gurobi, an exact branch-and-bound solver. The benefit of using Gurobi is that a bound on the optimality of the solutions is provided. Given that the objective function units are the same for all solvers, this optimality gap can be used for all solvers in this analysis. The runtime allocated to Gurobi was 24 hours per input to obtain good bounds for each instance.

D-Wave Hybrid Solver. As mentioned in Sec. 1.3, it is possible to construct algorithms which use QPUs in their inner loop, thus leveraging both classical and quantum resources. The smallest instance in our test set required 787 QUBO variables. While small for the SRP application, this is larger than could be solved on D-Wave QPUs at the time of experiments, which therefore necessitates the use of a hybrid algorithm in order to be solved with quantum annealing. We used a proprietary hybrid quantum-classical algorithm offered by D-Wave Systems, called the Hybrid Solver Service (HSS), which admits QUBOs with up to 10k binary variables. The HSS uses a QPU to optimize clusters of variables, allowing one to leverage the use of a quantum processor without the overhead of embedding. However, this hybrid algorithm does not allow direct access to control the QPU in its inner loop. Therefore, we consider the HSS as a black-box optimizer, and measure the performance as a function of the timeout parameter, similar to Gurobi and other black-box solvers.

We present the timing information allocated to each solver in Table 4.2, and the corresponding parameters in Table 4.3. For the D-Wave HSS, we limit the 30 and 50 shipment instances to only 5 minutes of runtime. We note that these 5 minutes were sufficient for the problems tested. Because we could not control the usage of the QPU in the D-Wave HSS, we report the QPU runtime in the timing results rather than a parameter. All software solvers were executed using single-threaded programs. To attempt a fair comparison, each QUBO solver was given roughly the same amount of time per test instance. However, the specific parameter choices corresponding to such times were found and set by hand.

We present the consolidated results from all solvers in Figure 4.2. While the total runtime of Gurobi was set to a 24 hour timeout (to obtain good lower bounds), good solutions with an optimality gap of less than 10 percent were already found after a few minutes for all instances. For the 30 and 50 shipment instances, we also obtained provably optimal solutions within the first few minutes of optimization. The solutions from Gurobi were significantly better than those obtained by solving

Instance	Simulated Annealing	Tabu	HSS
30	1 hr	1 hr	5 min (QPU: 3.0s)
50	1 hr	1 hr	5 min (QPU: 1.4s)
80	1 hr	1 hr	1 hr (QPU: 3.61s)
100	1 hr	1 hr	1 hr (QPU: 4.34s)

Table 4.2: Table of runtime allocated to each solver in the experimental setup.

Instance	Simulated Annealing	Tabu	HSS
30	2500 samples, 50000 sweeps	1 hr timeout	5 min timeout, $use\_qpu = True$
50	1600 samples, 50000 sweeps	1 hr timeout	5 min timeout, $use\_qpu = True$
80	1000 samples, 50000 sweeps	1 hr timeout	1 hr timeout, $use\_qpu = True$
100	500 samples, 50000 sweeps	1 hr timeout	1 hr timeout, $use\_qpu = True$

Table 4.3: Parameter sets used for each solver. Parameters not mentioned were set to default values.

the QUBO formulation. However, this is possibly due to both the fact that Gurobi is an exact solver and the way in which the MIP is discretized to form the QUBO, as explained in Section 4.1.3. The lack of discretization for Gurobi may result in more efficient packing of shipments along each edge, which would result in fewer trucks, and therefore a lower objective function value (truck km). Tabu search was able to find a near-optimal solution for the 30 shipment instance, but was unable to find even feasible solutions for any of the other instances. Simulated annealing was able to find feasible solutions, but only in the largest case of 100 shipments was the solution better than the direct shipment approach. The D-Wave HSS was able to find better-than-direct solutions for the 30, 50, and 80 shipment instances.

Throughout our initial experiments, we found that increasing the number of possible routes for each shipment does not directly correlate with improved solutions to the original problem (lower total truck km). This is due to the fact that each additional route creates more minima and a more rugged landscape. It is important to note that given the way we construct the QUBO– no trucks along an edge is a

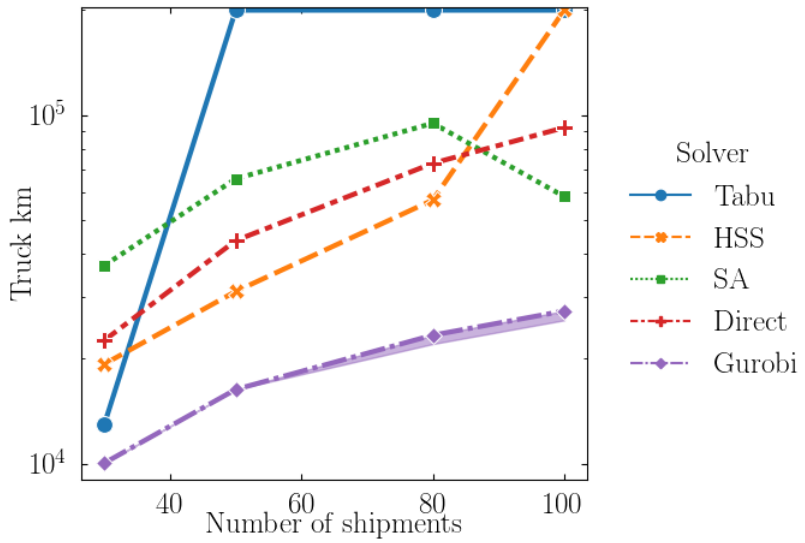

Figure 4.2: Performance of all solvers used in the experiments. We display the results in units of truck kilometers for ease of comparison. Simulated annealing (SA), Tabu, and the D-Wave HSS are QUBO solvers, Gurobi is a MIP solver, and the direct solutions are the simple baseline of one truck per shipment.

valid solution– increasing the number of possible routes can only create additional minima, not remove minima that have already been created. Furthermore, given that all constraints are implemented as penalty factors in the QUBO, this created an increased difficulty for the QUBO solvers to find optima. Given this insight, it is even more important to consider the number of QUBO terms (in Table 4.1) when modeling combinatorial optimization problems as QUBOs.

In general, both the SRP (in its original form) and the QUBO model the distance minimization of a simple objective function– total number of truck kilometers used to send shipments between nodes in a graph. The majority of our work focused on deriving methods to translate the MIP representation of SRP to a QUBO using both simple minimization objectives (truck kilometers as weights on decision variables), and hard constraints (packing constraints on edges in the graph) to test both quantum and classical optimization algorithms. In reality, it is evident that there is a significant amount of work required to find such valid QUBO representations for complex optimization problems inspired by real-world constraints. Despite the relatively straightforward description of the problem, correctly modeling the

solution landscape requires a more subtle approach, and required multiple iterations of derivations, as explained throughout the text. Of the algorithms tested, Gurobi performed the best despite being an exact branch-and-bound algorithm. Of the heuristics, we found that the D-Wave HSS was able to find better than greedy solutions for the smaller problem sizes tested. We stress that given our small test bed we cannot conclude any one solver being the best relative to the others, nor was this the intention. The context of the work presented was to assess the work that was required to transform a real-world optimization problem to a clearly-defined QUBO equivalent. We found that the bar defined as "acceptable" (finding solutions that are better than direct shipments) was surprisingly difficult for the heuristics to beat. This is important to note since simulated annealing was able to find valid solutions for all the problem sizes, but better-than-direct for only one problem (the largest). Furthermore, long runtimes were required to find these better-than-direct solutions, and yet were still far from optimality. The significance of this result is that despite QUBO being an NP-hard problem, the overhead in transforming any single optimization problem to QUBO may be detrimental to the performance of optimization algorithms which then solve the QUBO, and therefore it is not always worth the effort of the transformation. Furthermore, throughout the studies conducted in this section we found that naive and straightforward transformations to QUBO using known techniques is sometimes impractical. In particular, the inequality constraints required to correctly pack the shipment volumes in the truck capacities required such an increase in the number of variables, that additional data transformation was required to encode the constraint in the QUBO (discretization of the capacities).

4.3 Adapting real-world optimization problems to known QUBOs

We now turn our attention to a different approach to solving optimization problems. Here we perform the task of data reconstruction and classification using a QUBO model. Given some set of time series (TS) data, and a reference database, the task of reconstructing the candidate time series from features in the database is a hard problem. In our approach, we use a combinatorial optimization "oracle" in the form of a QUBO problem to model the task of the time series reconstruction, thus solving the underlying problem. In particular, we use the QUBO representation of the set

cover problem as the oracle performing this task. In contrast to the previous section, where the majority of the work was in deriving a valid QUBO, here we instead focus our work on finding methods to transform the data so that it can be solved effectively by the set cover problem without oversimplifying the original data. This is also in contrast to other quantum machine learning techniques, where it has been shown how to reformulate parts of classical classification algorithms as quantum subroutines that can be executed on error-corrected gate-model QPUs [101, 102, 103, 104]. In quantum annealing, similar numerical approaches have been shown in which the objective function of the classification task (minimizing distance metrics between high-dimensional vectors) has been directly translated to a QUBO, with each vector's possible assignment represented via one-hot encoding to physical qubits [105, 106].

By reformulating the critical task in our classification algorithm as a set cover problem, we introduce two novel ideas to quantum classification algorithms: (i) we avoid representing single vectors with polynomial numbers of qubits, instead representing the features within the data as the qubits, and (ii) we perform the classification task by transferring the core concepts of classification (and reconstruction) to the quantum algorithm for set cover, as opposed to a direct translation of a distance-based minimization procedure. This results in an algorithm that avoids a classical "learning" procedure, therefore requiring significantly fewer computational resources compared to other classical and quantum methods.

4.3.1 The set cover problem

The set cover problem is defined as follows: given a set of symbols (called a *universe*) $U = \{1, ..., n\}$, and a set of subsets V_i, such that $U = \bigcup_{i=1}^{N} V_i$, $V_i \subseteq U$, find the smallest number of subsets V_i whose union is U. This is a well-known NP-hard optimization problem, and is one of Karp's original 21 NP-complete problems [50]. There is a known QUBO formulation for the set cover problem provided in [77]. We start by defining the following binary variables:

$$x_i = \begin{cases} 1, & \text{if set } \ V_i \text{ is included,} \\ 0, & \text{otherwise.} \end{cases} \tag{4.11}$$

We let $\alpha \in U$ denote an element of the universe set, and m signify if element α appears in m subsets. Then, we have binary variables:

$$x_{\alpha,m} = \begin{cases} 1, & \text{if the number of } V_i \text{ which include } \alpha \text{ is } m, \\ 0, & \text{otherwise.} \end{cases} \tag{4.12}$$

We consider the full QUBO as a sum of two components:

$$H_A = A \sum_{\alpha=1}^{n} \left(1 - \sum_{m=1}^{N} x_{\alpha,m} \right)^2 + A \sum_{\alpha=1}^{n} \left(\sum_{m=1}^{N} m x_{\alpha,m} - \sum_{i:\alpha \in V_i} x_i \right)^2, \tag{4.13}$$

and

$$H_B = B \sum_{i=1}^{N} x_i. \tag{4.14}$$

The complete QUBO is given by $H = H_A + H_B$. The first summation in H_A imposes that exactly one of $x_{\alpha,m}$ must be selected in the minimum via a one-hot encoding. The second summation in H_A represents the number of times α is selected, and that this is equal to the number of selected subsets α appears in (m, as only one $x_{\alpha,m}$ can be 1 in the minimum). This is similar to the packing constraint from the previous section. The final term, H_B, serves to minimize the number of V_i needed to cover the universe U. The total number of variables required is $N + n(1 + M)$, where M is the maximal number of sets that contain given element of U. The limiting case where each element of V_i included covers only one element of U constrains the coefficient of H_A and H_B to $0 < B < A$. The closer the coefficients B and A, the more weight is given to (4.14), minimizing the number of elements selected from V.

4.3.2 Time series reconstruction as a QUBO

Classification techniques generally require specific data representation, similarity measure definitions, and algorithm selection. Similarly, in our QUBO approach, we represent the time series data as encoded strings from which we formulate semi-supervised classification and optimal reconstruction as a set cover problem, and provide metrics based on solutions to the set cover problem. While different than classical approaches [107, 108, 109, 110], we do not attempt to simplify the complexity of the problem, and introduce a method that is based on latent features within the data. The only assumptions we make about the time series data is that it is separated into two categories: a training set and a test set. The training set

we assume is labeled and is used as a reference database with which each time series in the test set is reconstructed from.

In order to reconstruct given time series data, we start by discretizing both the training and test data, and compare the encoded strings to generate the elements of our universe to form the set cover. This technique is crucial to allow feature-wise comparison of the data, as well as arbitrary reconstruction using existing (or training) data. There are many ways to discretize time series data, and exploring the trade-offs between the various methods is beyond the scope of this thesis. For our purposes, we use the symbolic Fourier approximation (SFA) method [111], as it provides differentiation between separate time series classes and features in high-dimensional data sets, allowing us to use these representative symbols for our set cover problem. Nevertheless, the exact discretization is data-dependent, with various hyperparameters (such as number of letters in the alphabet, length of each encoded string, etc.) present in the method. We therefore assume, under simple conditions, that we can treat the SFA method as a black-box that takes time series data as an input and returns symbolic strings encoding the data features as output.

Given the encoded strings, we introduce a pair-wise method to compare the time series features using what we call a "pulling procedure", illustrated in Figure 4.3. This pair-wise comparison is considered a pre-processing step necessary to formulate our set cover problem. Starting with one fixed string (red in the figure), we consider each encoded character as an independent element in the universe set[1] ($U = \{0, 1, 2, 3, 4\}$ in the figure). A second string (green in the figure) is compared element-wise by successively moving the second string along the first, as illustrated. At every iteration, all character matches between the two strings are recorded as a new set. In the example from Figure 4.3, the set of sets is $V = \{\{0\}, \{\emptyset\}, \{0, 2\}, \{\emptyset\}, \{1, 2, 3\}, \{\emptyset\}, \{\emptyset\}, \{3\}, \{\emptyset\}\}$.

The procedure is repeated for the rest of the encoded training tie series to form the set of sets V. In this setup, the SFA routine needs to be performed only once per time series, and that the pair-wise comparison is then performed in $O(n^2)$ time. The set which is a union of all subsets obtained via the pulling technique now represents the all features in common between the target (or test) time series

[1] It is important to note that by using the same encoding scheme for all time series data, we ensure that all string characters belong to the same alphabet.

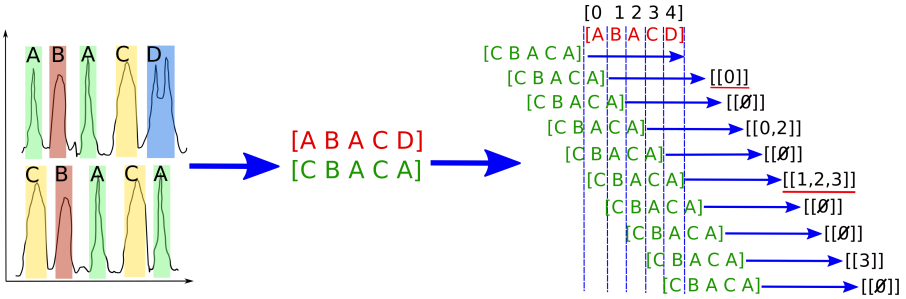

Figure 4.3: Schematic illustration of time series encoding and pulling procedure to produce subsets of set $V = \{\{0\}, \{\emptyset\}, \{0, 2\}, \{\emptyset\}, \{1, 2, 3\}, \{\emptyset\}, \{\emptyset\}, \{3\}, \{\emptyset\}\}$. The optimal selection to cover $U = \{0, 1, 2, 3, 4\}$ in this case would be underlined subsets $V = \{\{0\}, \{1, 2, 3\}\}$ with item numbers 0 and 4.

candidate and all other time series in the reference data set. Given this aggregate set, the goal is now to select the minimal subset that most closely reconstructs the universe, which is the NP-hard set cover problem. In other words, the task is to select the features in the reference database which correctly reconstruct the given test time series. In the case illustrated in Figure 4.3, the optimal selection of subsets is underlined in red. In principle, solutions of this set cover problem do not preserve order of elements, and allow the use of the same element multiple times. This feature is useful for time series comparison, as elements of the time series data can be permuted and duplicated without affecting our reconstruction method.

The final size of the set cover QUBO is heavily dependent on our choices during discretization. For example, the number of binary variables is equal to $N_{\text{train TS}}\,(2L-1)\,(L+1)$, where $N_{\text{train TS}}$ is the number of time series in the training set used for reconstruction, and L is the length of string that encodes the time series. Increasing the string length to encode each time series changes the size of the universe U. Allowing longer encoded strings to represent the data creates more subsets V_i. Therefore, there exists a trade-off between the granularity of the encoded strings and the ability to solve the set cover representation of the problem. Including more characters in our alphabet for discretization changes the non-empty sets V_i, which the number of quadratic elements in the QUBO depends on. The general trend is, however, that the number of the quadratic element decreases with the increase of the characters used in our alphabet. This is explained by the properties of the pulling procedure described above, since a smaller alphabet

produces more non-empty elements V_i which could be used for reconstruction of the universe U. In Figure 4.4 we show how varying these hyperparameters of the discretization affects the size of the QUBO problem, based on 20 test samples from one of our data sets used in experiments [112].

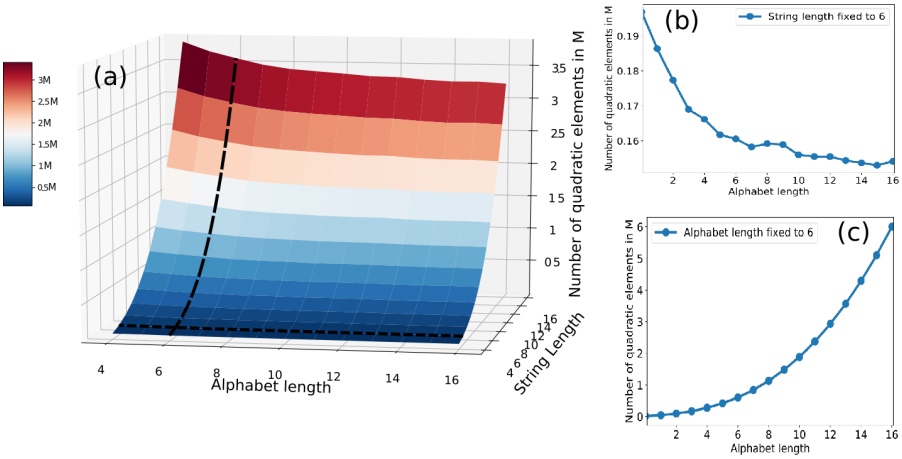

Figure 4.4: (a) The number of quadratic terms in millions as a function of string and alphabet length. (b) Quadratic elements as a function of alphabet length, with string length being fixed to 6. (c) Quadratic terms as function of string length, with alphabet length being fixed to 6. The corresponding isolines (b) and (c) are shown with dashed line on surface plot (a). Analysis was performed using 20 test samples from the BeetleFly data set [112].

4.4 Using QUBOs to perform classification

We can now combine the methods described in the previous sections– constructing the sets U, V from discretized data and the QUBO representation of the set cover problem– to perform semi-supervised classification from reconstructed data sets. In our case we use training data sets with known labels, and the task we solve is to use the labeled data to assign labels to the test set, given a valid reconstruction. Normally, the training set with labeled data is significantly smaller than unlabeled test set, which we exploit in our method.

We encode both the training and test data sets into strings using the pulling method described previously. We then perform the reconstruction procedure for

every time series in our test set using the entire training set. Each time series from the test set is assumed to individually form a universe U, and is to be reconstructed using the sets V_i, obtained via the pulling procedure. Explicitly, using Figure 4.3, the red string is the time series from the test data set, and all strings in the training set are pulled through (green strings) to obtain the V_i's. This allows us to compare every test time series to the full training set in one-versus-all manner. Then, using the universe U and V_i's from the pulling procedure, we formulate the set cover problem outlined in Section 4.3.1. Thus, a single solution to that set cover problem (even sub-optimal in the worst case) allows us to reconstruct each time series from the test set using a set of discretized features obtained from *all* elements which appear in the training set. Furthermore, since we employ metaheuristics to solve the set cover QUBO, various optima could yield different ways to reconstruct the test time series using the training set. Due to this, it is therefore the users' task to use these reconstructed strings to associate each test time series with a label from the training set.

To classify the reconstructed test time series data we evaluated three different similarity metrics using set cover solutions: largest common subset V_i, highest number of common subsets V_i, and largest sum of common elements in selected V_i. We briefly explain how each metric is calculated, and discuss the performance of each.

- **Largest common subset.** Given a candidate solution to the set cover problem, the label corresponding to the V_i which contains the most elements is selected. The label is then assigned to the test time series. This metric captures the longest continuous set of features from the training time series data, and assumes that is sufficient to determine the label.

- **Number of common subsets.** Frequently, multiple V_i's from the same training time series are used to reconstruct a test time series. In this metric, we count the number of V_i subsets used to cover the universe. The test label is assigned the same label as the training time series which appears most frequently in the set cover solution.

- **Largest sum of subsets.** This metric is a combination of the previous two. For every training time series that is used to reconstruct a test set, the total number of elements used by each is counted (summed over all V_i's). The

label which corresponds to the training time series with the largest sum is assigned to the test time series.

These metrics allow us to quantify the accuracy of our semi-supervised classification method. The first two metrics, being based on large sets of common features between the time series, performed the best (experiments and results shown in the next subsection). There was no significant difference between the two metrics, and the superiority of one metric over the other varied between data sets. The third metric, which was a combination of the first two, performed worse than either of the first metrics in the majority of the cases tested. While unexpected to begin with, this observation could be explained by the fact that because the third metric admits matches with many small subsets V_i that are selected in the set cover, this metric could miss significant signatures present in the time series data. Therefore, the largest common subset metric was selected for the experiments presented in the next section. It should also be noted that the use of labeled training data is not designed to not reach the accuracy of supervised learning methods. Moreover, there are modifications that could be made to the methods presented to improve the accuracy, for example increasing the word length and/or using a larger train set. Both are constrained in our use-case to prohibit excessively large QUBOs from being constructed. The goal of this method is to allow for relatively high accuracy using small sets of training data.

4.4.1 Classifying real-world time series data

We validate our method by using the set cover QUBO to reconstruct and classify a variety of open-source, real-world time series data. The benchmarking experiments performed here used labeled time series data available publicly [113, 114]. These data sets were used as-is in the experiments presented below. Validation was performed by measuring the classification rates of each methods on the labeled test data. We restricted the analysis to univariate time series data with two classes and small training set size. However, this method of semi-supervised classification can be used with any number of classes, at the cost of QUBO size. Since both the number of time series in the training data and the word length used to encode the data contribute to the number of variables in the QUBO, we select data sets that have small numbers of time series in the training set. The test and training sets used in these experiments are already determined and labeled by the source,

allowing us to easily calculate the classification rate of our method and avoid the step of selecting a training set. To benchmark the performance of our classification method, we compared the accuracy of our labeling method to semi-supervised and unsupervised classical classification methods. The results of these experiments for the various data sets are summarized in Table 4.5. To test the robustness of our method we collected a variety of data sources of different types. We briefly review each source and provide a literature reference for further details. We note that in the data sources' accompanying cited works, higher classification rates than our methods are reported using supervised algorithms. In this analysis we do not consider supervised classification algorithms, and instead compare our semi-supervised quantum-based approach to similar classical algorithms.

SonyAIBORobotSurface1 [115] data is sensor data collected from a small, dog-shaped, quadruped robot. It is equipped with multiple sensors, including a tri-axial accelerometer. In the experiments we classify between roll accelerometer measurements on two classes of surfaces: soft carpet and hard cement.

GunPoint [116] data includes motion tracking of actors' hands during gun-drawing and gun-pointing actions. For both classes the X-component of the actor's right hand centroid is tracked and used to distinguish between the two classes.

TwoLeadECG [117] and **ECG200** [117] are electrocardiogram data sets available at the PhysioNet database [118]. The first includes long-term measurements from the same patient using two different leads. The classification task aims to differentiate between each lead signal. In contrast, the second ECG200 set contains electrical activity recorded during one heartbeat. The two classes are the normal heartbeat and a Myocardial Infarction records.

BeetleFly [112] time-series data is generated from binary images developed for the testing of shape descriptors. The external contour of these images is extracted and mapped into the distance to the image center. The two image classes are contours of beetles and flies.

Chinatown [119] data is collected by an automated pedestrian counting system in the city of Melbourne, Australia. The classes are based on weekday or weekend traffic.

The QUBOs generated by our methods were too large to be embedded and optimized using the largest available QPUs (D-Wave 2000Q at the time of experiments). The exact sizes of the QUBOs for each data set are shown in Figure 4.5. To solve

the QUBOs we used the simulated thermal annealing metaheuristic which was used for previous experiments in this thesis. The specific implementation of SA was from the D-Wave Python package for classical QUBO optimizers [99]. We found that 20,000 samples and 1000 SA sweeps (with geometric interpolation of the inverse temperature) were sufficient to ensure that low-energy local minima were sampled within reasonable times per QUBO. We use the default SA settings in the package for initial and terminal inverse temperature selection (for more information about the implementation of SA we refer the reader to [99]).

The specific parameters used for the time series encoding to generate the QUBOs are shown in Table 4.4. In general, the longer the time series are and the fewer time series are in the training set, the finer the discretization method required to accurately classify the test data. In all data sets we were able to reconstruct each test time series with elements from the training set, as explained in Section 4.3.1. The distributions of the number of variables in each QUBO for all data sets is shown in Figure 4.5.

We provide an illustrative example of our QUBO-based reconstruction and classification in Figure 4.6 using the BeetleFly data set. The task is to reconstruct the data in Figure 4.6 (a) using (b) and (c). For this example, an alphabet of size 5 was used for encoding, color-coded in the figure. The results of the set cover problem, formulated using the methods explained in previous sections, are three sets, shown as v_1, v_2, and v_3 in Figure 4.6. Meaning, each box (representing a fifth of the time series data per box) that appears in one of the subsets forming the solution is designated as such. Specifically, v_1 = [‘A’, ‘E’], v_2 = [‘E’, ‘B’], and v_3 = [‘C’]. Therefore, the union $v_1 \bigcup v_2 \bigcup v_3 = U$, where U =‘ACEEB’, the test time series data to reconstruct. For classifying the reconstructed sample, we refer to the classes of the training data used for the reconstruction, and note that the training samples in Figure 4.6 (b) and (c) belong to two different classes. Using the similarity metrics defined above, it is easy to determine that v_1 and v_2 both originate from the time series (b), whereas only v_3 (which contains only a single element) is obtained from (c). Therefore, (a) is assigned the same label as (b). This example is representative of the majority of cases encountered during classification, with components of the reconstructed time series varying across multiple training samples, and often also across multiple classes.

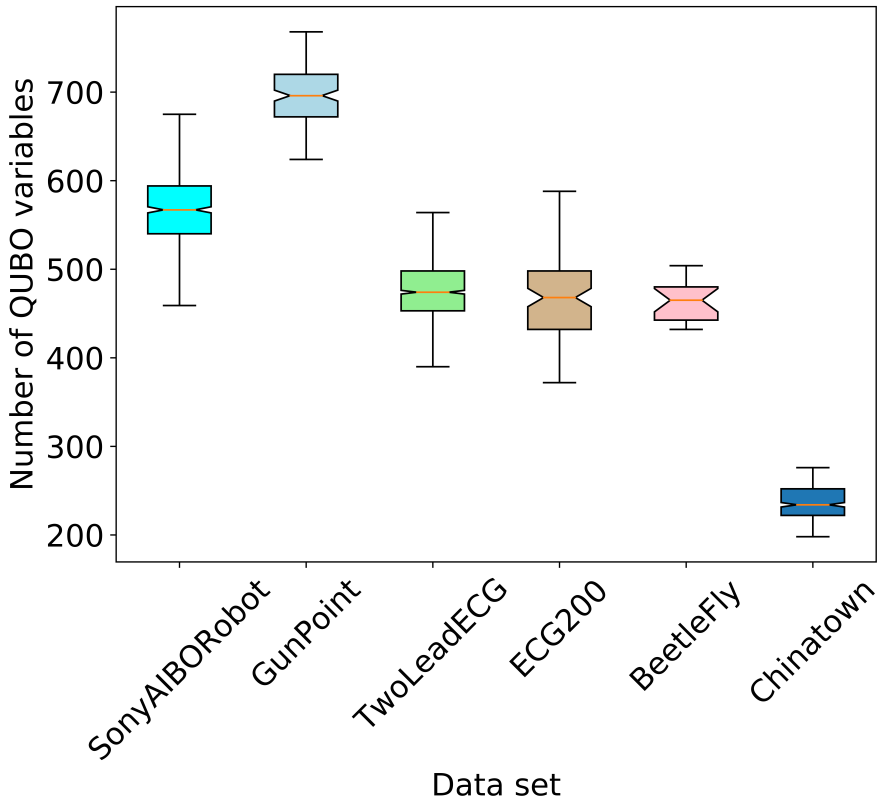

Figure 4.5: Distribution of number of QUBO variables for all data sets in Table 4.4.

4.4.2 Classification benchmarking

For the purposes of evaluating our QUBO-based classification method quantitatively, two classical time series classification algorithms were compared based on dynamical time wrapping (DTW) [120] measures: k-means classification and a classical analogue of the semi-supervised method described above. The motivation for using these specifically is that both are based on pair-wise similarity metrics as in the approach presented here. DTW applied to temporal sequences aligns the pair series in a non-linear way to minimize differences and calculate Euclidean distance afterwards. The DTW measure could be applied directly in unsupervised k-means classification or similarly to the method described here in the semi-supervised fashion. We use k-means classification with pairwise DTW metrics calculated

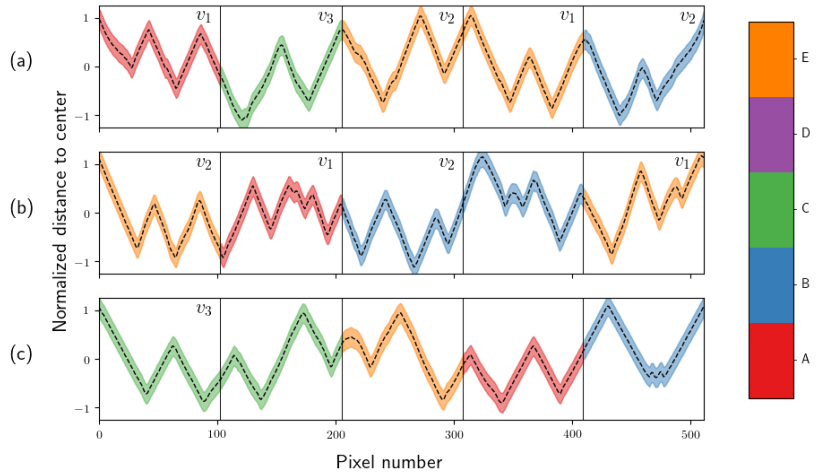

Figure 4.6: An illustrative example of reconstruction and classification from the BeetleFly data set. (a) A test time series sample (encoded as 'ACEEB') reconstructed from two training TS. Each box in the sub-figure is encoded as a single letter in a string, as per the color bar. The subsets v_i obtained from the pulling procedure and used to reconstruct this data are shown both in the reconstructed (test) time series and in the training time series. (b) The first training data used for reconstruction and classification (encoded as 'EABBE'). (c) A second time series used for reconstruction (encoded as 'CCEAB').

on the original TS (before encoding), with the labels being assigned based on belonging to one of two clusters. The second method assigns the test TS labels are by the DTW metric directly, calculated pairwise between each training and test TS (without encoding). We use these two methods to calculate classification rates for all data sources in the experiments.

As expected, the semi-supervised QUBO-based method outperforms classical unsupervised methods. We note however, that the QUBO-based method operates on a reduced dimensionality in contrast to the classical methods which use the original TS, where full information is preserved. Even under this consideration the accuracy of QUBO-based method is comparable with the semi-supervised DTW methods, and could be improved still by enriching the set V, i.e. by augmenting the training set or increasing the discretization granularity. The worst performance of the QUBO-based algorithm is observed on the TwoLeadECG data set. This

Data set	Data type	Train/Test size	Time series length	Word/Alphabet length
SonyAIBORobotSurface1	Sensor	10/601	70	8/8
GunPoint	Motion	30/150	150	5/5
TwoLeadECG	ECG	20/1139	82	5/5
ECG200	ECG	20/100	96	5/5
BeetleFly	Image	20/20	512	5/5
Chinatown	Traffic	20/345	24	5/5

Table 4.4: Table with data set description, number of time series in training and test sets, length of time series, and length of each encoded string and number of different letters used to encode data set.

could be explained by the nature of our method, as well as the sensitivity of the ECG data. By using the set cover problem, we allow for permutations of subsets of TS data in the reconstruction of the test TS. It is likely that this permutation of TS segments, and similar representation in Fourier space of the signals from the two leads in the ECG measurements, makes our method not suitable for this kind of data. The highest accuracy is obtained using the BeetleFly and Chinatown data sets. In the first case, many permutations of the training set to construct the test set are permissible, which our method takes advantage of. The accuracy of our method is additionally improved by the relative size of the training set, further augmenting the combinatorial space of permutations. This robustness can also be explained by the dimensionality reduction technique for this data set: the 2D BeetleFly images (with different orientations) were mapped to 1D series of distances to the image centre, which again is beneficial for permutation-based methods. The Chinatown data set, for comparison, contained significantly shorter TS than BeetleFly. Encoding the Chinatown TS data with the same word length as BeetleFly resulted in higher granularity representations, and ultimately higher accuracy. This provides additional evidence that the accuracy of our method can be improved by increasing the granularity of the encoding.

Among the advantages of our method is the utilization of significantly less data with respect to conventional classical methods, as well as a one-versus-all comparison that allows the selection of segments of data from multiple sources to reconstruct a single time series. This provides an additional robustness in the method with respect to permutations of time series segments during the reconstruction. In order to formulate this problem as a QUBO we apply time series dimensionality reduction by encoding each time series as a separate string. This encoding procedure and

Data set	QUBO method c1/c2/weighted	k-means c1/c2/weighted	DTW c1/c2/weighted
SonyAIBORobotSurface1	0.7/0.9/0.78	**0.85/0.97/0.92**	0.97/0.63/0.83
GunPoint	**0.76/0.79/0.78***	0.53/0.51/0.52	**0.82/0.77/0.79***
TwoLeadECG	0.6/0.62/0.61	0.65/0.7/0.68	**0.86/0.94/0.9**
ECG200	0.61/0.82/0.75	**0.62/0.8/0.79**	0.87/0.51/0.64
BeetleFly	**0.85/0.89/0.87**	0.64/0.83/0.73	0.62/1.0/0.82
Chinatown	0.72/0.91/0.86	0.37/0.78/0.67	**0.89/0.98/0.94**

Table 4.5: The classification accuracy measured on two classes and weighted average reported for QUBO-based and classical DTW-based methods. Bold text signifies the most effective classification method (based on the weighted average of the two classes) for each data set tested. Asterisk denotes a tie between the methods within statistical variance.

selection of comparison metrics define the hyperparameter space of the problem. The QUBO-based classification method performed the best on image and traffic data, which is consistent with our method's inherit ability to utilize permutations of features/data within the time series to perform reconstruction.

Time series reconstruction and classification has a wide variety of useful applications, such as: management of energy systems, factory process control, sensor systems, and many more. The methods introduced in this section show how to reformulate the tasks of reconstruction and classification of real-world data so they can be solved as QUBOs. This is a fundamental departure from the traditional methods used in solving optimization problems with QUBOs, and so we consider this a novel contribution to the field of optimization of real-world problems which can be built on in the future.