Optimal Control of Traffic Signals using Quantum Annealing

Hasham Hussain†‡ · Muhammad bin Javaid†‡ · Faisal Shah Khan† · Archismita Dalal§ · Aeysha Khalique§.

December 17, 2019

Abstract Quadratic Unconstrained Binary Optimization (QUBO) is the mathematical formalism for phrasing and solving a class of optimization problems that are combinatorial in nature. Due to their natural equivalence with the two dimensional Ising model for ferromagnetism in statistical mechanics, problems from the QUBO class can be solved on quantum annealing hardware. In this paper, we report a QUBO formatting of the problem of optimal control of time-dependent traffic signals on an artificial grid-structured road network so as to ease the flow of traffic, and the use of D-Wave Systems’ quantum annealer to solve it. Since current-generation D-Wave annealers have a limited number of qubits and limited inter-qubit connectivity, we adopt a hybrid (classical/quantum) approach to this problem. As traffic flow is a continuous and evolving phenomenon, we address this time-dependent problem by adopting a workflow to generate and solve multiple problem instances periodically.

1 Introduction

As predicted by Moore’s law [1], electronic computing technologies have reached the nano-meter scale and are well on their way to converging to the quantum scale within the next decade, meaning that electronic devices will have to be fabricated in a way that accounts for, and controls, quantum effects. Such devices will be instances of quantum electronics which, when assembled appropriately, will give rise to quantum circuits and hardware architectures that will constitute quantum computers. First generation quantum electronics and quantum computers are already available commercially in the form of quantum random number generators (IDquantique [2]), special purpose quantum annealers (D-wave Systems [3]) and limited universal quantum computers based on superconducting (Google [4]) and ion-trap (IonQ [5]) technology platforms.

The successful control of quantum features such as quantum entanglement and correlations can lead to quantum algorithms with incredible speed-ups in computational time. The famous results known as Grover’s [6] and Shor’s [7] algorithms respectively show that with a universal quantum computer, it is possible to search an unstructured database in time that is quadratic in the size of the database, and that an integer can be factored in time polynomial in the number of its

†H. Hussain and M. bin Javaid have contributed equally to this work
‡School of Natural Sciences, National University of Sciences and Technology, H-12 Islamabad, Pakistan
¶Center on Cyber Physical Systems and Department of Mathematics, Khalifa University, Abu Dhabi, UAE
§Institute for Quantum Science and Technology, University of Calgary, Alberta T2N 1N4, Canada
§National Centre for Physics (NCP), Shahdra Valley Road, Islamabad 44000, Pakistan
digits. This is in contrast to exponential time to perform each of these tasks on current “classical”
computers. Dramatic as these speed-ups are in theory, quantum computers that can successfully
implement these algorithms are predicted to be at least a decade away.

However, current first generation quantum annealers can be used to solve QUBO problems in
time that is, more often than not, less than the time it takes to solve them on classical hardware
and software optimizers [8,9]. We begin the discussion of our test case of optimal control of traffic
signals with a brief introduction to the QUBO class of problems and quantum annealing. A QUBO
problem deals with minimizing an unconstrained quadratic polynomial over binary variables,
\[ \min \sum_{i \leq j} Q_{ij} x_i x_j \]  

or equivalently,
\[ \text{Obj} = x^T Q x \] 

where Obj is the function to be minimized [10]. In equation (2), \( x \) is a vector containing \( N \) binary
variables, and \( Q \) is an \( N \times N \) matrix of real numbers describing the relationship between the vari-
ables. Given the matrix \( Q \), finding binary variable assignments to minimize the objective function
in this equation is equivalent to minimizing an Ising model, a well known problem in physics
which is known to belong to NP [11]. An important question to ask when converting an optimization
problem to a QUBO form is whether the problem may be reduced to a set of binary (yes/no) decisions.
Examples of real world problems that have been formulated as QUBOs may be found in [12,13,14].

D-Wave Systems produces quantum annealing devices. The quantum processing unit (QPU)
on D-Wave’s quantum annealer is designed specifically to solve problems from the QUBO class.
In this QPU, each qubit represents a variable, and couplers between qubits represent the costs
associated with the qubit pairings. The QPU is a physical implementation of an un-directed graph
with qubits as vertices and couplers as edges between them. A detailed breakdown of the quantum
annealing algorithm followed on the back-end by the D-Wave QPU may be found in [15].

In the following section, we briefly talk about the software package known as dwave-qbsolv.
This package provides both a classical algorithm for solving QUBO instances and an all-in-one
package for submitting problems to D-Wave’s quantum annealer over the Internet. Then we arrive
at the core of the paper: defining the problem of controlling traffic signals in a road network in such
a way so as to maximize traffic flow and minimize the total time wasted by cars waiting behind red
signals. We explain how to approach this rather open-ended problem mathematically as a series
of QUBO instances, cover the programming that was involved in this regard, and discuss the results
we obtained. Using these results, we also touch upon the potential advantage, in terms of scaling,
of using quantum annealing hardware to solve such a problem. All QUBO instances are solved on
the D-Wave QPU DW_2000Q_2.1 available via cloud service.

1.1 QBSolv

A detailed description of the working of the QBSolv algorithm can be found in [16], but briefly,
if we choose to solve a QUBO locally on our classical computer, we have the option of using the
software package dwave-qbsolv. It implements a tabu [17] search algorithm for finding the mini-
mum. Tabu search is a modified version of a local (neighborhood) search for finding the minimum
of a function. It takes a potential solution and checks its immediate neighbors in search of a better
solution.

Unlike other local search methods, tabu search can accept worsening moves if no improving
move is available, for example, when stuck in a local minimum - so tabu is a less ‘greedy’ algorithm
than traditional local search. Forbidden solutions (solutions already explored) are added in a ‘tabu
list’ so as to forbid the algorithm from checking those again.

The dwave-qbsolv package can send problem instances to the D-Wave QPU for solution as
well. For this, it adopts a “hybrid quantum/classical” approach which is characterized as follows.
For problems too large or with arbitrary connectivity that cannot be mapped directly onto the
QPU’s hardware architecture, it breaks the problem into sub-problems that can be mapped into the D-Wave QPU and solved piecewise. The breaking down process is carried out classically, and the resulting sub-problems are solved one by one on the D-Wave QPU using quantum annealing.

Next, we discuss our method of casting the problem of traffic signal optimization into a QUBO problem.

2 Optimal Control of Traffic Signals

The problem at hand is: how would one control traffic signals such that maximum traffic flow is achieved in minimum time? We made use of D-Wave’s LEAP service [18] to solve certain QUBO instances of this problem where needed, although for the most part during the development of this problem we used the classical tabu solver from the dwave-qbsolv package. The reason for this choice of action was that the time spent using the D-Wave QPU for problem instances was far too long for regular repetitive use as the program required. When using this service, much time is spent in latency and in the problem queue.

We point out here that while our current work is in the same spirit as the one by Neukart et al. in [14], our approach is quite different. In the work of Neukart et al., minimizing traffic congestion was achieved by assigning the most optimal route to each vehicle so as to minimize the overlapping of the routes in each vehicle’s itinerary with those in other vehicles’ itineraries. In our model of traffic flow, we do not consider any vehicle’s route; instead, we only use information about the number of vehicles behind every signal at any given moment in time, and the speed with which they will likely travel on any given road segment. This information is then used to formulate cost functions after regular time intervals. Each cost function, when minimized, returns the most optimal configuration of traffic signals at that particular time which will cause the flow of traffic to be as smooth as possible. In short, we look to minimize traffic congestion by finding the optimal way to control the traffic signals with respect to time.

2.1 Problem Description

We assume that the road network is a square grid with identical intersections where the road segments meet; each road segment is made up of two adjacent roads with traffic moving in opposite directions, as depicted in Figure 1.

![Fig. 1 The shape of the map](image)

As Figure 1 shows, we have defined six modes in which the traffic signals may be activated at any given intersection. We consider only those cases in which vehicles can go straight, turn right, or both.
Let the index on the intersections (nodes) of the map runs over $i$, and the index on the modes runs over $j$. Then we have binary variables $x_{ij}$ which represent mode $j$ on node $i$, and $x_{ij}$ is 1 if mode $j$ is active on node $i$ and is equal to 0 otherwise.

Now that we have defined the binary variable to be used in the QUBO model for this problem, let us consider what we are trying to optimize in this scenario by choosing modes at each intersection so that

1. the modes over the map activate in such a way so as to clear out the highest amount of stopped cars at any given time, and
2. the modes over the map activate in such a way so as to ensure that cars don’t have to stop at subsequent traffic lights. The idea is that once a car has been stopped at a red signal, upon it turning green, the car should not have to also stop at the next signal.

Note: There is a time-dependent aspect to this problem. Cars will keep moving and thus the locations and numbers of stopped cars will keep changing. Therefore, multiple QUBO instances will need to be formulated and solved repeatedly after fixed time intervals - say, 5-10 seconds. Each time a QUBO instance is solved, we obtain a new configuration of modes all over the map.

2.2 Cost Function

To achieve goal number 1 above, we need a cost term that provides a negative cost (thus, an incentive) to activate the mode at an intersection that would clear out the most traffic. We construct the following term

$$Q_1 = -\lambda_1 \sum_{i=1}^{n} \sum_{j=1}^{6} C_{ij} x_{ij}^2,$$

where $x_{ij}^2 = x_{ij}$ since it is a binary variable. $\lambda_1$ is a positive constant which we set equal to 1 for the purposes of our work. $C_{ij}$ represents the maximum number of cars that can be cleared out by activating mode $j$ on intersection $i$. For calculating $C_{ij}$, consider Figure 3.
Fig. 3 $a^i_k$ represents the number of cars standing in the indicated position on any node $i$ (excluding those cars that wish to turn left). $k$ runs from 1 to 4, representing each incoming lane at an intersection.

However, not all cars will wish to go straight - some may want to turn right. Consider then figure 4.

Fig. 4 $f^h_i$ represents the fraction of cars that want to keep traveling straight from their respective lane.

Therefore, we can calculate $C_{ij}$ as follows:

\[
C_{i1} = f^1_i a^1_i + f^2_i a^2_i \\
C_{i2} = a^2_i \\
C_{i3} = a^1_i \\
C_{i4} = f^3_i a^3_i + f^4_i a^4_i \\
C_{i5} = a^4_i \\
C_{i6} = a^3_i
\]

The cost of activating the mode at any intersection that will clear out the most amount of traffic will be the least, due to the negative sign in the expression. Therefore, this term fulfills its intended purpose. Of course, we will still need a constraint term that forbids all $x_{ij}$ being assigned a value of zero (no mode active at any intersection). We will introduce this constraint later. Let us first turn our attention to the term that will fulfill goal 2, namely

\[
Q_2 = -\lambda_2 \sum_{i=1}^{n} \sum_{j=1}^{6} C_{ij} x_{ij} [\tau_{i,a'} \lambda_3 C_{a',a} x_{a',a} + \tau_{i,b'} \lambda_4 C_{b',b} x_{b',b} + \tau_{i,c'} \lambda_5 C_{c',c} x_{c',c} + \tau_{i,d'} \lambda_6 C_{d',d} x_{d',d}],
\]

In equation (10), the $\tau_{ij'}$ terms are binary variables that deal with the synchronization of the traffic signals on subsequent intersections $i$ and $i'$ only at certain times. The $C_{ij}$ terms are analogous to those in equation (3). The $\lambda_2$, $\lambda_3$, and $\lambda_6$ terms are constant factors. The indices are elaborated by Figure 5 and the functions of the various terms are detailed below.

Consider an intersection $i$ with mode $j$. We need to look at the corresponding intersections being influenced by this mode on intersection $i$. The objective is to find modes on the affected intersections such that a smooth flow of traffic is achieved. The intersections being influenced depend on which mode is active on intersection $i$. 
The highlighted modes shown in figure 5 (at their respective intersections) receive a greater negative cost (bonus) for being selected. This is achieved by setting $\lambda'_3 > \lambda_3$ (we chose the values of 0.7 and 0.3 respectively). In particular, these are the modes in which traffic is allowed to go both straight and right. $\lambda'_3$ and $\lambda_3$ may be set equal as well, or given a number of different values depending on the situation we are attempting to model.

What we have done so far is merely synchronize the modes of nearby intersections, preferring modes in which both right-going and straight-going traffic is allowed to pass. However, there is a problem: modes synchronizing in these intersections should not be instantaneous. They should synchronize only when a car that set out from $i$ reaches the next connected intersection.

Recall that we are solving multiple QUBO instances repeatedly. We know the length of a road segment and the speed limit on it (assume speed limit of segment = speed of average car on that segment). Therefore we know how long the average car will take to travel between any two intersections ($\Delta t_{ii'}$). We also know the current time $t$ that has elapsed. Using this information, we construct parameters $\tau_{ii'}$ which have value zero except when enough time has elapsed for a car to reach the next intersection under consideration, say $i'$, from an intersection $i$. In this case, $\tau_{ii'}$ is set equal to 1 for that QUBO instance. This corresponds to a bonus (negative cost, equal to $\lambda_3$) being added to the objective function if it synchronizes the modes of those two intersections. Mathematically, we can represent this condition as

$$t \mod \Delta t_{ii'} \approx 0$$ (11)

where $\tau_{ii'}$ equals 1 only if this condition is satisfied. For cases where $t < \Delta t_{ii'}$, the condition

$$\Delta t_{ii'} - t \approx 0$$ (12)

is used instead. These parameters $\tau_{ii'}$ ensure that mode-synchronizing only occurs when a car has had enough time to move from intersection $i$ to $i'$ - otherwise, $\tau_{ii'}$ is zero and the synchronizing terms vanish.
Fig. 6 For example: the average car takes 4 seconds to cross this segment; so the mode-synchronizing should occur only when the car is sufficiently close to the signal at the end of the segment.

We also have $C_{ij}$ terms which contain traffic density information in this expression. As elaborated previously, a term $C_{ij}$ simply tells the number of cars that would be allowed to move if mode $j$ were active on intersection $i$. This is to ensure that if there is a conflict between mutually exclusive synchronizing options, the option that would benefit the highest number of vehicles is chosen.

Finally, we have a constraint term that ensures that one mode, and only one, is selected at each intersection, giving

$$Q_3 = \lambda_4 \sum_i [1 - \sum_{j=1}^6 x_{ij}]^2,$$

(13)

where $\lambda_4$ is the penalty for violating the constraint. It should be assigned a value large enough so as to always be satisfied in the minimum of the QUBO. Notice that for any fixed $i$, the value of this term is zero only if exactly one of the $x_{ij}$ for that given $i$ is chosen to be equal to 1. Therefore, in order to avoid adding a massive penalty cost to the Objective function, the solver will be forced to choose a solution where only one mode is selected for each intersection. The full objective function for this problem, $(Q_1 + Q_2 + Q_3)$, is therefore

$$\text{Obj} = -\lambda_1 \sum_{i=1}^n \sum_{j=1}^6 C_{ij} x_{ij}^2 - \lambda_2 \sum_{i=1}^n \sum_{j=1}^6 C_{ij} x_{ij} [\tau_{i,a'} \lambda_3 C_{a'} a x_{a',a} + \tau_{i,b'} \lambda_3 C_{b'} b x_{b',b} + \tau_{i,c'} \lambda_3 C_{c'} c x_{c',c} + \tau_{i,d'} \lambda_3 C_{d'} d x_{d',d}] + \lambda_4 \sum_i [1 - \sum_{j=1}^6 x_{ij}]^2$$

(14)

$\lambda_1$, $\lambda_2$, and $\lambda_4$ are the weightages given to their respective cost function terms. Increasing $\lambda_2$, will, for example, increase the priority of mode synchronization.

In the next section, we will cover the details of how we programmed this cost function. We also made a traffic simulation which used this cost function to decide the states of the traffic signals on a map. We compared the results of using this QUBO to find optimal traffic signals, with both fixed-cycle signals and signals that effectively only used the first and last terms of the Objective function (so, no signal synchronization). This was to see whether there was actually a benefit to synchronizing signals in this manner or not. We also compare the performance and results of the classical QBSolv tabu solver vs the D-Wave QPU.

3 Programming

We made a program that constructs QUBO instances (Q matrices) for this problem using the following data.
• Map of traffic signals. This map contains signals at intersections and the distances between connected intersections.
• Speed limit of each road segment (needed for calculating average time taken to travel between any two intersections)
• Current traffic density on the segments leading to an intersection (the $a^k_i$ values). Cars are counted as contributing to the traffic density as long as the distance between successive cars is below a certain number - for example, 5 meters.

We now have enough information to start constructing the Q matrix from the objective function of this problem. When the program runs, it executes as follows.

• Generate a graph representing a grid map, resembling that in Figure 6. This map contains the lengths of each segment, and the speed limit of each segment. We then calculate the time taken for the average car to traverse this segment using these two quantities (assume speed limit of that segment = speed of the average car on that segment). At the end of this process, we have a quantity for each segment that equals the average time needed to traverse it.
• Using the current traffic density on the road segments behind signals ($a^k_i$), compute all $C_{ij}$ values as outlined in the problem formulation.
• Compute the objective function associated to a Q matrix.
• Solve this QUBO instance using dwave-qbsolv (has the option of either solving using tabu search method or by sending to D-Wave’s quantum annealer)
• Interpret the resulting vector of binary variables ($x_{ij}$) in terms of modes at every intersection.

Now comes the time-dependent part. We have only solved using the first term (and, of course, the constraint) of the objective function so far. What about the second term that synchronizes linked intersections after certain times?

In order to incorporate that term, we constructed a simulation that places cars on the map according to the initial traffic density data information. Once a single QUBO instance has been solved according to the first and the last terms of the cost function [Eq. (14)], it takes the resulting modes at each intersection and moves the cars (according to the speed limit of each segment). It functions in iterations - and we may speed up or slow down the simulation by assigning a different value for the real-time a single iteration is supposed to represent. Once this simulation has been kick-started, we can start making use of the second term in the objective function. The simulation ‘knows’ how much time has elapsed since its starting, and it also has the information of the time taken to traverse any segment in the map. Using this information, it keeps calculating all $\tau_{\alpha\beta}$ all over the map according to the aforementioned condition for $\tau_{\alpha\beta}$.

The simulation has the additional purpose of updating the traffic density information for the next time a QUBO instance is solved. In short, the simulation has three purposes.

• Calculate $\tau_{\alpha\beta}$ to be used in the objective function for the next time a QUBO instance is generated and solved.
• Move the traffic - resulting in changed $C_{ij}$ for the next time a QUBO instance is generated and solved.
• Visualize the moving traffic on the map.

It is not necessary that a QUBO instance be solved every time a single simulation iteration completes, since a single simulation iteration only represents 1 second in simulation time. We may set a QUBO instance to be solved after, for example, every 5 seconds in simulation time, which means after every 5 iterations. For iterations where a new QUBO is not solved, the traffic will keep on moving as per the current configuration of signal modes all over the map. When it is time to change the modes, the program will pick up the current $C_{ij}$ values from the simulation and the current $\tau_{\alpha\beta}$ and compute the Q matrix. When this new QUBO instance is solved, the modes at each intersection will change and as a result certain cars on the map will need to stop, start moving, or keep moving as before - the simulation handles this.

The flowchart in Figure 7 outlines the entire process followed by the program.
Some details about the simulations we ran are as follows.

- Number of intersections = 36 (6×6 grid map)
- Number of cars = 4320 (spread evenly across the map at first - cars keep moving / trickling out of the 6×6 grid as time passes)
- Possible speed limits of roads (in meters/second) = 11, 17, 22, 28 (speed limits were randomly assigned to each road segment out of these)
- Length of each road segment = 1km
- \( f^i_k = 0.7 \) - at every intersection, we have assumed 70 percent of incoming cars from any given direction wish to keep traveling straight, as opposed to turning right
- Total number of iterations run in every case = 150 (150 seconds, or 2.5 minutes, in simulation-time)
- \( \lambda_1 = 1, \lambda_2 = 60, \lambda_3 = 0.3, \lambda_4 = 0.7, \lambda_4' = 0 \) (there are many possible \( \lambda_2 \) and \( \lambda_4 \) values which will also produce optimal results. As a rough guideline, both of these should be at least as large as the largest coefficient in the term \( Q_1 \))
- A new QUBO instance was formulated and solved after every 5 iterations (5 seconds in simulation-time). So, the traffic signals over the map changed after every 5 seconds.

3.1 Summary of the workflow

The objective of the controlling traffic signals problem is to maximize the flow of traffic. That is, we are tasked with controlling the traffic signals in a road network over a period of time (which lights to manipulate). In order to do so, we have the following (classical) information available.

- Road network map with traffic lights on the intersections.
- Distances between intersections and the speed limits of these segments.
- Number of cars stopped / moving bumper-to-bumper behind a traffic signal.
- The current time elapsed since the simulation began.

Now, in order to solve this problem, we divide the problem into a chronological workflow:

1. Process map data, including the lengths of all segments and their respective speed limits. (Classical)
2. Process traffic density data and information as to which signals to synchronize in this instance (if any). (Classical)
3. Formulate the QUBO matrix. (Classical)
4. Find a solution that provides smoothest flow of traffic across the route. (Hybrid classical/quantum if D-Wave QPU used)
5. According to the resulting signals configuration, the simulation moves traffic on each segment resulting in updated traffic density data. The simulation also figures out which signals to synchronize for the next run. (Classical)
6. Repeat steps 2-5.

Here, ‘Classical’ refers to calculations carried out on classical machines.

4 Results and Observations

4.1 Quality of Solutions

We require some kind of objective metric using which we can quantify the optimality of a certain method of controlling traffic signals. The metric that seems most reasonable would be the total amount of time spent waiting behind red lights by all cars, but weighted by the speed limit of the road each car will travel to after the light turns green. So, for example, a car stopped behind a highway road has its wasted time weighted more, since it could potentially travel further in the same time were it allowed to move. This formula is:

\[
\text{Time Wasted} = \frac{\text{speed limit of next road segment}}{\text{maximum speed limit}} \times N
\]  

(15)

where N is the number of cars wanting to move on to the next road segment. This metric is calculated on each intersection during every iteration, and then summed continuously as the simulation proceeds. The bar chart in Figure 8 illustrates the metrics for three cases calculated over 150 seconds in simulation-time each. The three cases are: full cost function, no synchronization \((Q_2 = 0)\), and fixed signal cycles, where signals are changed purely based on time elapsed and not on the traffic data. All three of these cases shown below were run classically (using QBSolv’s tabu solver)

![Fig. 8 Comparison of three cases when solved using tabu solver](image)

When run on the D-Wave QPU, the full cost function case \((Q_1 + Q_2 + Q_3)\) returned a metric of 31.28 hours wasted. The other two cases were not tested on the D-Wave QPU. Solution quality on the QPU can be improved by manually increasing the anneal time [15] - the default is 20 µs. It should be noted that the classical tabu search algorithm we used is completely reproducible, and produces exactly the same solutions (and thus metric) every time it is run for a certain problem.
4.2 Time Scaling of Solutions

A QUBO instance was solved after every 5 iterations and the time taken to solve it measured. Below are plots of the time taken for every solution instance for both quantum (QPU) and classical (tabu solver) cases:

**Fig. 9** Time taken in solving traffic signals QUBO on D-Wave QPU. Note that the actual time taken on the QPU for each instance is lower than that implied by the graph, since this time inevitably includes the internet latency and queuing time, which can both fluctuate considerably.

**Fig. 10** Time taken in solving traffic signals QUBO using QBSolv’s tabu solver

Note that there is no general large-scale upward or downward trend in the solution time as we go through all the iterations, in both the QPU and classical cases. The fluctuations in the graph may be attributed to a number of factors, such as queue time, internet latency, processor scheduling and so on.

The QPU times according to the graphs above were all taken by measuring the amount of time taken to return a solution from the QPU, and as such includes the time taken to process QUBO instances and split them into sub-problems for the QPU and embed them. This procedure is outlined in [16].
5 Conclusions

We have shown that the real-world optimization problem of optimally controlling traffic signals can be formulated so as to be solvable using quantum annealing. In particular, we have shown that there is a marginal benefit to synchronizing subsequent traffic lights by including the $Q_2$ term in the cost function, so this potentially justifies formulating the problem as a QUBO - as the function of the first cost function term $Q_1$ is so simple so as not to require being formulated as a QUBO problem. This benefit of signal synchronization can be increased significantly if the parameters of the problem, such as the $\lambda$ values in the cost function, are fine-tuned according to the situation and only a certain subset of traffic signals are considered for synchronization.

Naturally, the question arises as to whether it is practically feasible to solve such problems using quantum annealing - in particular, is there a time advantage to using quantum annealing instead of classical computers? Time is of the essence in many optimization problems - for example, in the traffic signals optimization problem, it is absolutely vital that we obtain an answer for a QUBO instance every 5-10 seconds in the scenario that this algorithm is being used in a real-world application.

Even though quantum annealing is outperformed in many cases by classical algorithms such as tabu search and simulated annealing, if we have dedicated access to D-Wave’s QPU minus latency and problem wait time, it may be feasible for solving large instances of time critical problems like this one. Why is quantum annealing not clearly faster, though, when anneal time on D-Wave’s device remains fixed regardless of problem size? The reasons are:

- The connectivity between qubits in the physical QPU graph is limited; it is a rather sparse graph. Thus it is not possible to embed large problems directly in one go - as minor embeddings often don’t exist for the full large problem due to the sparsity of the QPU graph. So a large problem must be split up into smaller chunks and embedded one by one and solved piecewise on the QPU. This is an issue - because one ‘chunk’ takes approximately the same time to go through annealing as a large problem would if it were possible to embed it. So, a lot of time is being wasted in splitting up the problem, finding minor embeddings for each and then going through the anneal and resampling cycle for each chunk. This would not be an issue if the QPU graph was densely connected - then it would be possible to directly embed large problems in one go and solve.
- Limited number of qubits - real-world problems often have several thousand variables or much more, and this combined with the aforementioned lack of connectivity between qubits leads to the same issue as before - inability to directly embed large problems in one go.
- Error rates in D-Wave’s quantum annealing devices are high [15] - the qubits are ‘noisy’ due to problems in manufacturing, problems in shielding, and so on. As a result, there is a large probability of qubit states collapsing prematurely due to perturbations. Due to this, a large number of resampling cycles needs to be run for each problem instance - adding more time required for the computation.

However, all three parameters (qubit connectivity, number of qubits, error rates) are improving with each new generation of annealers. The number of qubits have consistently gone up - from 128 qubits in 2012 to 2000 qubits in 2017. A 5000+ qubit version is slated for release in mid-2020, with vastly improved qubit connectivity - 15 connections per qubit compared to 6 as in the current version [19]. It is likely that in the not-so-distant future, quantum annealing will surpass classical algorithms and be suitable for solving very large problem instances that take too long to solve classically using even the best available classical algorithms.

Acknowledgements We acknowledge D-Wave quantum annealer facility to run our code and helpful discussion with B.C. Sanders.

References

1. G. Moore, Electronics 38(8) (1965)
2. IDQuantique, URL https://marketing.idquantique.com/acton/attachment/11868/f-0227/1/-/-/QuantumRNG_WhitePaper.pdf
3. D. Systems, URL https://docs.dwavesys.com/docs/latest/c_gs_2.html
4. A.K.B.R.e.a. Arute, F., Nature pp. 505–510 (2019)
5. B.K.D.S.e.a. Wright, K., Nature Communications (10) (2019)
6. L. Grover, Proceedings, 28th Annual ACM Symposium on the Theory of Computing p. 212 (1996)
7. P. Shor, Proceedings 35th Annual Symposium on Foundations of Computer Science. IEEE Comput. Soc. Press.
8. C.C. McGeoch, Adiabatic Quantum Computation and Quantum Annealing: Theory and Practice (Morgan & Claypool Publishers, 2014)
9. C.C. McGeoch, C. Wang, in Proceedings of the ACM International Conference on Computing Frontiers (ACM, New York, NY, USA, 2013), CF ’13, pp. 23:1–23:11. DOI 10.1145/2482767.2482797. URL http://doi.acm.org/10.1145/2482767.2482797
10. F. Glover, A. Kochenberger, arXiv:1811.11538 (2018)
11. F. Barahona, Journal of Physics A Mathematical General 15(10), 3241 (1982). DOI 10.1088/0305-4470/15/10/028
12. M. Hernandez, M. Aramon, Quantum Information Processing 16(5), 133 (2017). DOI 10.1007/s11128-017-1586-y. URL https://doi.org/10.1007/s11128-017-1586-y
13. R.Y. Li, R. Di Felice, R. Rohs, D.A. Lidar, npj Quantum Information 4, 14 (2018). DOI 10.1038/s41534-018-0060-8. URL https://doi.org/10.1038/s41534-018-0060-8
14. F. Neukart, C. Compostella, C. Seidel, D. von Dollen, S. Yarkoni, B. Parney, Frontiers in ICT 4, 29 (2017). DOI 10.3389/fict.2017.00029. URL https://www.frontiersin.org/article/10.3389/fict.2017.00029
15. S.E. Venegas-Andraca, W. Cruz-Santos, C. McGeoch, M. Lanzagorta, Contemporary Physics 59(2), 174 (2018). DOI 10.1080/00107514.2018.1450720. URL https://doi.org/10.1080/00107514.2018.1450720
16. M. Booth, S.P. Reinhardt, A. Roy, https://www.dwavesys.com/resources/publications, (2017)
17. F. Glover, Computers & Operations Research 13, 533 ("1986")
18. D-wave: The quantum computing company. www.dwavesys.com/take-leap Accessed: 2018-09-30
19. D-wave previews next-generation quantum computing platform-d-wave systems. www.dwavesys.com/press-releases/d-wave-previews-next-generation-quantum-computing-platform Accessed: 2018-09-30