



OPEN

## Supply chain logistics with quantum and classical annealing algorithms

Sean J. Weinberg<sup>1</sup>, Fabio Sanches<sup>1</sup>, Takanori Ide<sup>2</sup>, Kazumitsu Kamiya<sup>3</sup> & Randall Correll<sup>1</sup>✉

Noisy intermediate-scale quantum (NISQ) hardware is almost universally incompatible with full-scale optimization problems of practical importance which can have many variables and unwieldy objective functions. As a consequence, there is a growing body of literature that tests quantum algorithms on miniaturized versions of problems that arise in an operations research setting. Rather than taking this approach, we investigate a problem of substantial commercial value, multi-truck vehicle routing for supply chain logistics, at the scale used by a corporation in their operations. Such a problem is too complex to be fully embedded on any near-term quantum hardware or simulator; we avoid confronting this challenge by taking a hybrid workflow approach: we iteratively assign routes for trucks by generating a new binary optimization problem instance one truck at a time. Each instance has ~ 2500 quadratic binary variables, putting it in a range that is feasible for NISQ quantum computing, especially quantum annealing hardware. We test our methods using simulated annealing and the D-Wave Hybrid solver as a place-holder in wait of quantum hardware developments. After feeding the vehicle routes suggested by these runs into a highly realistic classical supply chain simulation, we find excellent performance for the full supply chain. Our work gives a set of techniques that can be adopted in contexts beyond vehicle routing to apply NISQ devices in a hybrid fashion to large-scale problems of commercial interest.

Quantum algorithms have the capacity to offer enormous performance improvements over known classical algorithms for solving important problems like integer factoring and quantum mechanical simulation<sup>1–3</sup>. However, despite extraordinary effort and investment, the current state of quantum hardware remains too immature for there to be practical computational value from any quantum device that exists today. Recently, calculations have been performed with a quantum computer that outperform classical computing<sup>4</sup>, but these calculations do not solve a problem of practical use.

Given these realities, much research has been devoted to heuristic algorithms, often of a hybrid classical-quantum nature, that aim to offer computational advantage even with problematic hardware<sup>5–9</sup>. While such algorithms that are suitable for the noisy intermediate-scale quantum era (NISQ)<sup>9</sup> are not proven to offer complexity advantages over classical methods, they have potential to become an oasis during the long road to the development of large-scale fault-tolerant quantum computers.

NISQ algorithms and rapid development of quantum hardware has caught the attention of numerous industries that are in search of solutions that can improve efficiency in their operations. To test the potential for NISQ algorithms in a given use case, it is commonplace to begin with a complex problem and to then identify a miniaturized version of the problem which can then be attacked with a NISQ quantum algorithm. For example, many problems that arise in operations research can be recast as a quadratic unconstrained binary optimization (QUBO) problem. Such problems can be downsized and then solved using the quantum approximate optimization algorithm<sup>5</sup> (with simulated or real hardware) or by execution on quantum annealing devices like those produced by D-Wave Systems<sup>10</sup>.

There is no question that such studies are of value, but there are also major issues with them. Problem instances must be reduced to very small toy models that barely reflect the true nature of practical problems. Circuit model quantum algorithms cannot currently be tested, even on simulators, much beyond 30 binary variables. Quantum annealers can become cumbersome around a few hundred variables due to embedding challenges. Private companies may not be inclined to fund quantum computing studies when restricted to such small use cases and when the quality of solution and run time is all but guaranteed to be easily outperformed by classical algorithms with readily available hardware.

<sup>1</sup>QC Ware Corp., Palo Alto, CA, USA. <sup>2</sup>Aisin Corporation, Tokyo Research Center, Chiyoda-ku, Tokyo, Japan. <sup>3</sup>Aisin Technical Center of America, San Jose, CA, USA. ✉email: randall.correll@qcware.com

In this work, we develop an alternative approach to downsizing problems for commercial studies. We begin with a realistic vehicle routing problem that arises in a company's operations. Rather than distilling the mathematical problem and constructing a version with very few variables, we instead use a hybrid workflow approach: we iteratively construct small QUBO instances that are of reasonable size for near-term hardware with NISQ algorithms. Solutions to these small instances do not solve the full and very complex vehicle routing problem, but they do provide a route for a single truck. We use that route to update the remaining unfulfilled demand and we then repeat the procedure. After obtaining solutions for all trucks, we input the routes into a highly realistic simulation of the flow of all trucks and carried boxes, taking into account various constraints that are difficult to include in QUBO instances. The final result is a viable solution to the full-scale routing problem which can be compared to solutions obtained through other means.

Using iterative approaches to build up a heuristic solution to a complex problem instance is certainly not a new idea. However, the construction we give is carefully crafted to suit the needs of near-term quantum computing hardware: the problem instances at each step are QUBO instances with no more than a few thousand variables. These QUBO instances will be appropriate for circuit model quantum hardware with a few thousand noisy qubits. In the more near term, the quantum annealing algorithm might be more promising than circuit model algorithms. Quantum annealing attempts to use quantum tunneling through energy landscapes to more efficiently reach global low-energy states that represent the minimum of an optimization problem<sup>11–15</sup>. And D-Wave Systems offers quantum annealing hardware that can optimize QUBO instances with thousands of variables already.

However, our instances are still too large for direct application on D-Wave annealers due to the difficulty of embedding QUBO connectivity graphs. To address such problems at scale, our approach decomposes the overall problem into sub-problems that can be optimized separately. Solving the parts of the decomposed problem cannot guarantee a global solution, but our motivation is to target a portion of the problem that we can address with quantum annealing at this time. The supply chain logistics problem involves three parts: vehicle routing, multiple vehicle coordination, and pick-up/drop-off requirements. We forgo a integrated multi-vehicle approach and focus on single-vehicle approach with pick-up and drop-off requirements subject to schedule constraints as the main problem to be optimized. In particular, we use a commercially available hybrid quantum annealer and classical solver that attempts to boost the performance of a classical solver by including information from a quantum solver<sup>16,17</sup>.

To build up a multi-truck solution, we iterate through solving single-truck solutions, updating the remaining demand, solving for an additional truck, and repeat until a stopping condition is met. This provides us with a list of suggested routes for all the trucks. And finally, we relax some of the simplifying assumptions and use a simple but now more detailed classical (non-quantum) simulation to rectify the final multi-vehicle solution.

The construction described in this work yields a viable quantum-classical solution for a specific problem of substantial commercial value. Specifically, we study the problem of routing trucks in the supply chain of Aisin Corporation, a Japanese automotive component manufacturer. We emphasize, however, that the broad approach of deploying quantum algorithms on small problem instances to build up a hybrid solution to full-scale commercial applications is important well outside of the context of supply chain optimization, and in fact such approaches may be the first ways that quantum computing will be used outside of a research setting.

**Outline.** In “Vehicle routing problems”, we explain the vehicle routing problem (VRP) that models the Aisin Corporation supply chain. There are many ingredients to the routing problem, so we gradually build it up by starting with a well-known vehicle routing problem and adding various ingredients in a step-by-step fashion. “Single truck PUBO” describes the structure of binary optimization problem instances that we construct for each truck in an iterative fashion. These are the optimization problems that can be submitted to quantum hardware in a quantum implementation of our methods. In “Supply chain workflow”, we describe the algorithm for iteratively constructing the optimization problems from “Single truck PUBO”, and using their solutions to update initial conditions related to the remaining unfulfilled demand. In “Truck loop annealing runs”, we execute our workflow. As a place-holder for full quantum solutions to the binary optimization problem instances, we use simulated annealing and the D-Wave hybrid tool provided by D-Wave Systems. These solution methods offer insight into how the workflow might behave when a direct quantum algorithmic approach becomes feasible. Finally, in “Full-scale simulation”, we input the route solutions from our simulated annealing and D-Wave hybrid runs into a full-scale supply chain simulation which tracks, in detail, the locations of all boxes and trucks, ensuring that realistic and cumbersome constraints are satisfied. This allows us to evaluate the overall performance of our methods by determining the percentage of demand that is fulfilled given the number of trucks used.

## Vehicle routing problems

Although there are many aspects of supply chains that can be optimized, we focus on vehicle routing. Mathematically, the wide variety of combinatorial optimization problems go by the umbrella name of “vehicle routing problems” (VRP)<sup>18,19</sup>. The VRP variant studied in this paper is closely modeled after the supply chain of Aisin Corporation, and is thus very complex. It involves many trucks, nodes, and goods that must be carried by trucks in ways that satisfy certain routing constraints. We explain this vehicle routing problem by building it up, starting from a very basic one.

**Basic vehicle routing problem.** The VRP discussed in this section is most similar to the split-delivery capacitated vehicle routing problem, but since it's our base example from which we will build up a more realistic supply chain model, we refer to it informally as the “basic VRP”. An instance of the basic VRP is specified by:

1. A graph  $G$  with  $n + 1$  nodes. For convenience, the nodes are labelled as  $z_0, z_1, \dots, z_n$
2. One special node  $z_0$  selected from the graph which is called the *depot*.
3. An  $(n + 1) \times (n + 1)$  matrix  $T$  with nonnegative entries and  $T_{ii} = 0$  for all  $i \in \{0, 1, \dots, n\}$  called the *time matrix*.
4. A nonnegative number  $d_i$  assigned to each node  $z_i$  except for the depot ( $i \neq 0$ ). These numbers are called *initial demands*.

The nodes of the graph can be abstract, but in many cases they are explicitly given as coordinates for locations that trucks might drive to. The time matrix entry  $T_{ij}$  is supposed to be the time it takes a truck to drive from node  $i$  to node  $j$ . That is why we require  $T_{ii} = 0$ . The initial demand  $d_i$  is supposed to be the amount of material that must be carried by a truck from the node  $z_i$  to the depot node  $z_0$ .

Here the “amount of material” is intentionally vague. In a practical application, demand can be quantified by volume, by weight, or even by monetary value. For our purposes, we will use volume as the standard meaning for “amount of material” which makes it sensible that trucks have a limited carrying capacity.

A candidate solution to this VRP is given by a route: a list of integers  $\xi = \xi_1, \xi_2, \dots, \xi_k$  where  $k$  is some positive integer (called the route length) and each  $\xi_j$  is an element of  $\{0, 1, \dots, n\}$ . We require that  $\xi_1 = 0$  and  $\xi_k = 0$  so that the truck starts and ends at the depot. Given such a sequence, there are two questions:

- What is the total driving time for  $\xi$ ?
- Is  $\xi$  a demand-satisfying route?

The driving time for the route  $\xi$  is computed in the obvious way:

$$\text{time}(\xi) = \sum_{t=1}^{k-1} T_{\xi_t \xi_{t+1}}. \quad (1)$$

The question of whether or not the route is demand-satisfying requires some further elaboration. In short, a route is demand satisfying if it will result in a truck carrying all of the initial demand to the depot. The mathematical description of this concept is somewhat inelegant due to the nature of picking up material from one location and bringing it to another. The truck which starts at  $\xi_1$  and follows the route has a *capacity* which we always take to be 1. This is the amount of demand that the truck can store. The amount of demand the truck is carrying at a given time is called *on-board demand* and the truck's on-board demand is initially zero, and we denote this by  $e_1 = 0$ . (The index 1 on  $e_1$  refers to the fact that it is the on-board demand after stopping at  $\xi_1 = 0$  which is the first time step.) Then, the truck drives to  $\xi_2$  and picks up demand from  $\xi_2$ . The amount picked up is the largest allowed amount:

$$e_2 = \min(1, d_{\xi_2}) = \min(1, e_1 + d_{\xi_2}).$$

Note that this is either the full demand at node  $\xi_2$  or it is a portion of that demand that fills up the truck to capacity. After this step, we update the *off-board* demand located at node  $\xi_2$ . This means that we change  $d_{\xi_2}$  by reducing it by the amount that the truck took from  $\xi_2$ :

$$d_{\xi_2}^2 = d_{\xi_2} - (e_2 - e_1).$$

Explicitly writing  $e_2 - e_1$  even though  $e_1 = 0$  is pedantic, but it helps clarify how this is generalized: the off-board demand at each time step is calculated by starting with the off-board demand from the prior time step and reducing it by the amount by which the on-board demand increased. The only other ingredient to understand the flow of demand in our VRP is that when the truck returns to the depot node at some time step, it drops off all demand there. The on-board demand returns to zero, and off-board demands are not changed.

To be explicit, the general update rules of on-board and off-board demand are

$$e_t = \begin{cases} 0 & \text{if } \xi_t = 0 \\ \min(1, e_{t-1} + d_{\xi_t}^{t-1}) & \text{otherwise} \end{cases}$$

and

$$d_j^t = \begin{cases} d_j^{t-1} & \text{if } \xi_t = 0 \\ d_j^{t-1} - (e_t - e_{t-1})\delta_{j,\xi_t} & \text{otherwise} \end{cases}$$

respectively. In these equations,  $t$  is a time step index and we are defining  $d_j^{t=0}$  to be the demand  $d_j$ . Moreover,  $\delta_{kl}$  is the Kronecker delta.

With the definitions above, we can state that the optimization goal of this basic vehicle routing problem is to find, among all demand-satisfying routes, the one which completes its route in the smallest time. Here, time is defined as in Eq. (1).

**Tensor demand structure.** The basic VRP discussed in “[Basic vehicle routing problem](#)” has the property that all demand must be taken to the same destination node (the depot). This is built into the mathematical

description of the problem because the off-board demand has a *vector demand structure*. This means that the demand at a given time step  $t$  is given by a vector

$$\mathbf{d}^t = (d_1^t, d_2^t, \dots, d_n^t).$$

In a realistic supply chain, we do not have the luxury of a single delivery destination. Goods from a given node may be split into groups which need to be taken to various delivery destination nodes. To accommodate such a situation, we introduce the concept of a *tensor demand structure*. We begin with a rank-2 tensor.

*Rank-2 demand.* Consider a graph with  $n$  nodes  $z_1, \dots, z_n$ . (There is no longer any need for a special  $z_0$  node). We introduce an  $n \times n$  matrix  $D^{t=0}$  with nonnegative entries. The meaning of the entry  $D_{ij}^0$  is, intuitively, the initial amount of demand that is located at node  $i$  and must be shipped to node  $j$ .

When the truck arrives at node  $i$  at time  $t$  with this sort of demand structure, a new issue arises: a *pickup selection* decision must be made. There are  $n$  different sorts of demand that can be picked up from node  $i$ :  $(D_{i1}^{t-1}, D_{i2}^{t-1}, \dots, D_{in}^{t-1})$ . Because of this difficult issue, we cannot simply determine whether or not candidate solution giving only a route without an explanation of how we perform pickup selection is in fact a demand-satisfying solution.

Assuming that pickup selection can be accomplished in a reasonable fashion, we can obtain a new matrix  $D^t$  by reducing  $D^{t-1}$  by the amount of demand picked up by the truck from node  $i$ . However, now a second issue appears: now that demand is on the truck, we have to remember which parts of the on-board demand must go to which destination nodes. This can be dealt with by promoting on-board demand at time  $t$  to a vector  $E^t = (E_1^t, E_2^t, \dots, E_n^t)$ . The meaning of  $E_i^t$  is that, after the operations at time step  $t$  (including pickup),  $E_i^t$  is the amount of demand on the truck which must be delivered to node  $i$ .

Now that there is on-board demand in the truck, we need to revisit what happens when the truck first arrives at a given node  $i$ . Before pickup selection or any other operation, the first step is now to completely drop off demand  $E_i^{t-1}$ . Mathematically, this simply means setting  $E_i^t = 0$ . If we wish, we can also keep track of the overall total demand satisfied after each time step, in which case we would iteratively define a sequence  $S$  by

$$\begin{aligned} S^0 &= 0 \\ S^t &= S^{t-1} + E_{\xi_t}^{t-1} \end{aligned}$$

where, as in “[Basic vehicle routing problem](#)”,  $\xi_t$  refers to the node visited at time step  $t$ .

*Arbitrary rank demand.* The ideas of a demand matrix  $D_{ij}^t$  can readily be generalized to higher-rank tensors. The reason for doing this is that in a practical supply chain (including the one our study is based on), there are delivery requirements along the lines of “move this box from node 3 to node 7, and then from node 7 to node 5”. Such multi-leg requirements may sound odd, but they can arise for numerous practical reasons. There may be capacity limitations at node 5, and node 7 may be a storage warehouse. Or perhaps the box needs to have an operation performed on it before its final delivery. Another important reason for multi-leg delivery requirements is that a cargo container may need to be sent somewhere else after delivery.

Whatever the reason, promoting the matrix and vector structure of  $D$  and  $E$  to higher rank tensors allows us to encode the data that we need for this new situation. An initial demand tensor  $D_{ijk}^0$  can be interpreted as “there is initially demand  $D_{ijk}^0$  located at node  $i$  which needs to first travel to node  $j$  and then travel to node  $k$ .”

Unfortunately, with higher-rank tensor structure like this, the operations that are performed when a truck arrives at a node become even more complicated. Consider first an empty truck arriving at node  $i$  at time  $t$ . It starts by performing pickup selection to decide what off-board to pick up: any part of  $D_{ijk}^0$  is fine as long as the first index is  $i$ . After the pickup, the off-board demand is correspondingly reduced. However, the loaded demand is now material with instructions like “go to node  $j$ , then go to node  $k$ ” so we must introduce a matrix on-board demand  $E_{jk}^t$  to track this. However, now that the truck has this on-board demand, when it later drives to node  $j$ , the on-board  $E_{jk}^t$  will be dropped off. This demand is *not satisfied* because it hasn’t reached its final destination of node  $k$ . We are therefore forced to introduce a rank-2 matrix off-board demand structure when this demand is dropped off! When that matrix off-board demand is later picked up, it is converted to rank-1 vector on-board demand.

In conclusion, rank- $r$  off-board demand will automatically require tracking off-board demands with ranks 2 through  $r$  as well as on-board demands with ranks 1 through  $r - 1$ . These can be separately tracked by a collection of tensors like

$$\begin{array}{c} D^{rt} \\ D^{r-1t} \quad E^{r-1t} \\ \vdots \\ D^2t \quad E^2t \\ D^1t \end{array}$$

or, alternatively, we can “embed” lower rank demand within a single higher rank demand tensor. Regardless of the organizational approach, there is no question that bookkeeping is one of the major issues that arise when dealing with this more realistic version of a vehicle routing problem.

As with the cases above, we can introduce a “total demand satisfied” sequence  $S^t$  which accumulates only when demand is sent to its final destination. We do not accumulate  $S$  when rank-2 on-board demand arrives at a node, but we do accumulate it when rank-1 on-board demand arrives because that node is the final destination for that material.

**Multiple trucks.** The next complexity to consider is the involvement of multiple trucks. This is intuitive and easy to describe mathematically, but it adds immense difficulty for optimization. The main observation to make about the mathematical structure is that there is an on-board demand for every truck, but there is only one off-board demand. Thus, we need a new index  $m$ , which ranges from 1 to the number of trucks  $N$ , added to on-board demand. For instance:

$$E_{m,ij}^t$$

for rank-2 on-board demand. We can allow different trucks to have different capacities  $C_1, \dots, C_N$ , but throughout our work we assume that all trucks have capacity 1.

The introduction of multiple trucks adds profound subtleties to the problem. The optimal solution to an instance with many trucks may involve highly collaborative relationships between trucks. Finding nearly optimal solutions may thus require an exploration of numerous heuristic optimization algorithms or machine learning approaches.

In our study, we take an approach that will not offer a truly collaborative solution, but may provide good solutions with excellent runtime. We find truck routes in an iterative fashion, starting with a single truck, updating the off-board demand expected to be satisfied by that truck, and then adding a second truck with the new demands. Repeating this process allows us to take advantage of some of the benefit of including many trucks, but without dealing with the most intractable aspects of this problem.

*Optimization goals with multiple trucks.* When dealing with multiple trucks, there is some ambiguity on the optimization goal for a routing problem. There are two reasonable goals to consider.

1. If the number of trucks is fixed and given, then the optimization goal is to minimize the total driving time among all trucks to satisfy all demand.
2. If the number of trucks is not fixed, then the optimization goal is to minimize total driving time for all trucks and to find the number of trucks that attains the lowest minimal total driving time.

In other words, we can either fix or optimize over the number of trucks.

**Individual boxes and the box soup simplification.** Another factor in the realistic vehicle routing problem that we are building toward is the fact that boxes are not abstract “material” but physical boxes with specific volumes and weights and specific routing requirements. Each box has a starting node and a list of nodes that it must arrive at before going to its final destination node. In other words, for every box  $a$ , where

$$a \in \{1, \dots, \text{number of boxes}\},$$

there is a corresponding list of nodes  $R_a$  which box  $a$  is required to visit. The number of elements in  $R_a$  is some integer  $r_a$  which is greater than or equal to 2. We sometimes call  $r_a$  the rank of box  $a$ . In other words, box  $a$  follows the path

$$R_a = (R_a^1, R_a^2, \dots, R_a^{r_a}). \quad (2)$$

There is also a volume for the box  $a$  which we denote as  $V_a$ .

Rather than dealing with all of this detail, we can simplify the problem by finding all boxes with the same required route  $R_a$ . We can put all such boxes together into a *box group*. For the optimization algorithm, there is no need to distinguish boxes within the same box group unless they have different volumes which will fill off-board and on-board demand differently. However, we can perform a *box group soup* simplification where we combine all of the boxes within the same group together and regard their total volumes as continuous. The box soup then determines the initial off-board demand tensor(s), but it plays no other role after that. In this case, the final solution we find may not be physically possible, but at least we won’t have to deal with the enormous complexity of tracking individual boxes.

**Restricted driving windows.** Typical VRPs involve minimizing driving time to accomplish the goal of fulfilling all deliveries. However, in a commercial setting there is a limitation on the time in which trucks can drive. We may fail to satisfy all demand, especially with heuristic algorithms.

Suppose that all trucks are only allowed to drive during an overall period of time  $T_{\max}$ . The trucks drive simultaneously during this time. Then, we are not necessarily guaranteed that it is possible to fully satisfy demand within that constraint.

In this situation, the objective function is no longer obvious. One possibility is to minimize driving time and maximize satisfied demand with some relative weighting. There is, however, a different commercially natural objective to consider. To minimize cost, it is important to minimize the number of trucks required. Thus, a useful objective is to find the smallest number of trucks such that it is possible to satisfy all demand.

**Aisin Corporation vehicle routing problem.** With all of the ideas above, we are finally in a position to describe the commercial routing optimization problem that we aim to address with an optimization workflow. For brevity, we will refer to the most general (and potentially complicated) of these commercial routing problems as the “logistical routing problem” (LRP).

The LRP is based closely on the supply chain of Aisin Corporation, a Japanese automotive manufacturing company. Their operations involve the delivery of parts between numerous facilities. Driving times between these facilities range from minutes to hours, and tens of thousands of boxes must be shipped by trucks between these locations on a daily basis. The Aisin Corporation routing challenge can be thought of as an instance (or class of instances) of the LRP. For example, the highest rank of demand tensors is 3. When referring specifically to the Aisin Corporation restricted version of the LRP, we will use the terminology “Aisin logistical routing problem” or ALRP. The ALRP is the instance for which we describe an implementation of our workflow in “[Execution and performance](#)”.

We strongly emphasize that neither the LRP nor the ALRP are meant to be rigorously defined computational problems. There is some ambiguity on the objective goal in these problems. The following is a summary of the features of the LRP and ALRP.

- *Nodes:* The number of nodes are arbitrary for the LRP. In the ALRP, there are 23 nodes.
- *Trucks:* Multiple trucks are allowed in the LRP and ALRP. For the ALRP, the typical number of trucks is in the 50–100 range.
- *Initial demand:* Off-board demand for the LRP can consist of an arbitrary (finite) number of demand tensors with rank greater than or equal to 2. The ALRP only has rank 2 and rank 3 off-board demand.
- *Driving window:* Driving windows are optional for the LRP. In the ALRP, the driving window is fixed (and, in reality, is 16 hours broken into two shifts).
- *Boxes:* For the LRP, demand can be broken into individual boxes. This is the case for the ALRP.
- *Time matrix:* The time matrix is arbitrary for the LRP. The ALRP time matrix is based on actual estimated driving times between facilities in Japan. Times range from minutes to hours in this case.
- *Objective:* The objective for the LRP/ALRP is ambiguous but is roughly to maximize demand satisfied while minimizing total driving time (which includes minimizing the number of trucks).

### Binary optimization and annealing

The annealing algorithms that we discuss below are heuristic algorithms that attempt to find the minimum or maximum of polynomials with binary variables. Many problems, especially discrete optimization problems, can be cast as polynomials of binary variables<sup>20</sup>. Routing problems like vehicle routing problems are not manifestly similar to optimizing a polynomial objective function, but we will be able to find polynomial optimization instances with solutions that can be used to build up approximate solutions for the realistic vehicle routing problem discussed in “[Aisin Corporation vehicle routing problem](#)”.

**Polynomials with binary variables.** By a “polynomial with  $n$  binary variables” we mean a polynomial from  $\{0, 1\}^n \rightarrow \mathbb{R}$  with real coefficients. While we are choosing the domain of the  $n$  variables to be  $\{0, 1\}^n$  there are other reasonable conventions like  $\{1, -1\}^n$ . In the former case, we sometimes say that the variables are *boolean* and in the latter case the variables are said to be *spin* variables. When we use the term *binary*, without clarification, we mean boolean. As an example,  $f : \{0, 1\}^2 \rightarrow \mathbb{R}$  defined by  $f(x, y) = 3xy - y$  is a polynomial with two boolean variables.

The computational problem PUBO, which we take to stand for *Polynomial Unconstrained Binary Optimization*, is the problem of finding the minimizing input to a given polynomial with binary variables. Similarly, the computational problem QUBO (*Quadratic Unconstrained Binary Optimization*) is the same as PUBO except that the polynomial has at most degree 2. PUBO (and thus also QUBO) are NP-hard, as can be seen from the fact that many NP-hard problems can be reduced to QUBO<sup>21</sup>. For the sake of brevity, we often use the term “QUBO” to refer to a quadratic polynomial with binary variables rather than the computational problem itself as in “ $f(x, y, z) = z + xy - x$  is a QUBO and  $g(x, y, z) = xyz$  is a PUBO”.

**Quantum and classical optimization algorithms.** There has been substantial interest in the QUBO and PUBO problem classes from the quantum computing community because various heuristic quantum optimization algorithms are designed for polynomials with binary variables. The quantum annealers of D-Wave Systems specifically solve QUBO instances. Simulated annealing, which is a class of classical algorithms that are somewhat analogous to the operation of quantum annealing, is well-suited for both QUBO and PUBO instances.

There are also very notable circuit-model algorithms that optimize PUBO instances. The quantum approximate optimization algorithm (QAOA)<sup>5</sup> is a variational algorithm that begins with a uniformly distributed quantum state and gradually evolves the state into one that, when measured, yields an approximate solution to the original PUBO. In certain respects, QAOA is a natural choice for our workflow once hardware evolves to the scale where around 1000 useful qubits, either error-corrected or sufficiently low-noise, can be manipulated in a circuit-model quantum computer. In lieu of this, the tests performed in “[Execution and performance](#)” exclusively use annealing algorithms which we now briefly review.

**Simulated annealing.** Simulated annealing<sup>22,23</sup> refers to a class of heuristic optimization algorithms that are motivated by the idea of starting in a “high temperature ensemble” and gradually “cooling” until a state finds its way into the minimum of an objective function. To be more concrete, consider a finite set of points  $A$  and an

objective function  $f : A \rightarrow \mathbf{R}$ . Assume that  $f$  is bounded below so that there is some  $a_0 \in A$  such that  $f(a_0) \leq f(a)$  for all  $a \in A$ . In addition, there is another important piece of structure needed for the set of points  $A$ : we need a concept of “neighboring” points. Assume, therefore, that for every  $a \in A$ , there is a collection of “neighbors” of  $a$ ,  $N(a)$ . While there are no formal requirements on the properties of  $N(a)$ , simulated annealing works best when  $N(a)$  tends to satisfy some things that are reasonable for so-called neighbors: we want all of the neighbors of  $a$  to have similar objective values to that of  $a$ . We also want the size of  $N(a)$  to be quite small compared to the size of  $A$ , and we also need to ensure that the elements of  $N(a)$  are efficiently determined given  $a$ .

PUBO and QUBO fit naturally into this framework. We set  $A = \{0, 1\}^n$ ; neighbors are naturally defined by putting a limitation on Hamming distance. For example, given a binary string  $x = (x_1, \dots, x_n)$ , we can let the  $n$  strings obtained by flipping a single bit of  $x$  be the elements of  $N(x)$ .

The final input needed for simulated annealing is a “cooling schedule” which is a value of a “temperature”  $T$  for every time step.  $T$  should be taken to start off as a larger number and gradually decrease to zero. We start in an initial guess  $a_1 \in A$  and look at all neighbors of  $a_1$ . We evaluate the objective function  $f(a_1)$  and compare it to  $f(a)$  for  $a \in N(a_1)$  and we select a new state  $a_2$  based on probabilities given by relative Boltzmann factors  $e^{-f(a)/T}$  for the various states. When  $T$  is large compared to objective function differences, each step randomly changes states with no consideration for value of  $f$ , but as  $T$  tends to zero, states evolve in an increasingly greedy fashion.

**Quantum annealing.** Quantum annealing is a quantum analog of the classical simulated annealing described above. And importantly, quantum annealing is the paradigm implemented on the D-Wave Systems quantum annealing computer, which was used in this study to find the solutions to the logistics routing problem instances. Quantum annealing is meant as a form of adiabatic quantum computing, where an initial quantum state is subjected to a time-dependent Hamiltonian in such a way that the state remains in the (approximate) ground state of the Hamiltonian during evolution. By evolving from a initial “driving” Hamiltonian to a “problem Hamiltonian”, one that represents the binary optimization problem instance that we want to solve, measurements of the final ground state yield solutions to the binary optimization problem instance<sup>24</sup>.

In more detail, a transverse field is applied to an ensemble of spin qubits to provide a high-energy superposition of states in an initial driving Hamiltonian,  $H_{\text{init}}$ . The transverse field serves as an effective high temperature. The transverse field is slowly lowered, while simultaneously a problem Hamiltonian,  $H_{\text{final}}$  is applied to the qubits via spin-spin couplings and external fields with increasing strength according to an annealing schedule

$$H(s) = (1 - s)H_{\text{init}}(s) + sH_{\text{final}}, \quad (3)$$

where a real number,  $s \in [0, 1]$ , is the annealing schedule parameter.

If done adiabatically—that is, sufficiently slowly—this allows the final alignment of the qubits, spin up or down, to represent a low-energy minimum configuration that corresponds to a solution to the problem Hamiltonian that was applied. The rate of annealing can be considerably slow, depending on the minimum gap between energy levels of the ground state and neighboring higher-energy states, and can lead to anneal times longer than the coherence time of quantum annealing devices. Adding to this coherence time limit the fact that other noise sources exist, fully adiabatic quantum computing has not yet been achieved.

Quantum annealing is a similar approach to adiabatic quantum computing that uses the similar anneal schedule and achieves low-energy solution states without strict adiabaticity<sup>6</sup>. The intuition behind quantum annealing is that during the anneal quantum effects will allow the exploration of energy structure of the problem by tunneling through energy barriers<sup>6,7,10,25,26</sup>. In practice, of course, noise limitations limit the effectiveness of the technique, but nonetheless, such devices have been built that increasingly are able to find low-energy states of problem Hamiltonians for ever increasing problem sizes.

**D-wave systems quantum annealing computer.** D-Wave Systems has been offering quantum annealing computers for commercial use for over a decade. Their latest model, the D-Wave Advantage, offers 5000 superconducting qubits operating at cryogenic temperatures. It is accessible via the Internet and is fully programmable to represent any problem in QUBO formulation of binary variables as described earlier in this section. In order to overcome the limitation of problem size—a limitation for all existing quantum computing devices of today—D-Wave Systems include a hybrid solver that uses the quantum annealing computer to provide promising starting points to large-scale classical computers. This allows researchers to work on the quantum annealing algorithm while running much larger problem sizes of practical interest. This hybrid solver was in fact used in this study to address a much larger and more realistic logistics problem than could be handled by just the quantum annealer alone.

**D-wave quantum annealer.** The important characteristics of a quantum annealer are the number of qubits, the amount of connectivity among the qubits, and the noise qualities of the qubits. The latest D-Wave Advantage also offers an increase in connections between qubits to 16, up from 6 on previous versions. This increase in connectivity allows more efficient embedding of logical problems onto the hardware graph of qubits, and thus enables larger problem sizes to be embedded onto the processor. The embedding of the logical problem onto the hardware qubit graph is important. One should recognize that the 5000-qubit processor cannot handle a problem of 5000 binary variables. The embedding requires multiple hardware qubits to be programmed as a logical node to represent each logical variable. For a fully-connected logical problem, one in which every binary variable interacts with all the others, one can only embed such a fully-connected problem of approximately 180 logical binary variables on the 5000 hardware qubits. Many problems of practical interest are not fully-connected logically, so larger problem sizes of hundreds of binary variables can often be embedded. However, to handle much larger problems of thousands of binary variables requires the use of the hybrid solver—quantum annealer and

classical heuristic solvers working together. Documents fully describing the characteristics and operating modes of the D-Wave Advantage quantum annealing computer can be found on their web site<sup>16,17,27</sup>.

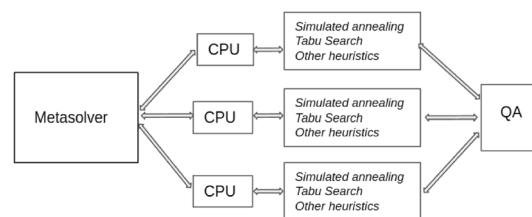
The performance of the D-Wave Systems quantum annealers has increased with each new generation of machine. While the latest machine still can not perform better than the best classical algorithms on multiple CPU/GPU compute hardware, they are narrowing the gap between their annealing quantum processor unit (QPU) and a classical CPU for certain problem types<sup>26</sup>. It is difficult to compare the performance of quantum annealing and classical heuristics in theory. Predictions of adiabatic quantum computing have some theoretical underpinning, quantum annealing itself does not. Additionally, in empirical comparisons, the quantum annealers are mostly limited by noise and other imperfections in their current state of technology, while classical algorithms and hardware are very mature. Research on improving quantum annealing is ongoing both from a qubit technology perspective and from an operational perspective. A case in point for the latter is exploring tailored annealing schedules different than the nominal linear annealing schedule shown above in Eq. (3). For example, inhomogeneous annealing schedules for each qubit can be applied and have shown significant improvements in obtaining higher probabilities for low-energy solutions<sup>28</sup>.

**D-wave hybrid solver.** The D-Wave Systems hybrid solver aims to bring quantum computing to bear on larger problem sizes than it can alone handle at this point in its maturity. The key idea is to use the quantum annealer to better guide a classical heuristic algorithm. Heuristic algorithms for optimization lie between the two extremes of exhaustive search and random sampling. Exhaustive search is, of course, too time consuming for large problems, scaling exponentially with problem size for combinatorial optimization. Random sampling is fast but not very good at finding extrema. Heuristics extend from each of these extremes. Examples are search techniques that can smartly eliminate large portions of the search space, such as branch and bound, and probabilistic sampling along with local search, such as in simulated annealing described above. For discrete optimization, which cannot benefit from the power of gradient descent, alternative heuristics have been developed that can help guide the search space or eliminate redundancy, such as tabu search, which excises previously explored territories of the search strategy.

Quantum annealing brings something additional into play. It is believed, as identified in the introduction, that quantum annealing will be better at exploring the global search space and, via the power of quantum tunneling, might avoid getting stuck in local minima. The D-Wave Systems hybrid approach invokes several of these approaches in an overarching meta-heuristic<sup>17</sup>.

A top-level functional diagram of the hybrid solver is shown in Fig. 1. The Metasolver on the left governs the overarching algorithm. The problem Hamiltonian in QUBO form is provided along with the stopping conditions, usually just an overall target run-time. The Metasolver launches multiple threads which uses classical heuristic algorithms running on CPUs or GPUs. These heuristic algorithms use versions of simulated annealing, tabu local searches, and additional proprietary heuristics. Simultaneously, the Metasolver uses the quantum annealer to search for promising solutions of smaller subsets of the problem, and these are fed back into the algorithm flow to provide additional promising starting points to the heuristic algorithms in each thread. The Metasolver collects a set of best solutions until the stopping condition is reached, and the results are reported back to the user.

In performance testing on benchmark optimization problems, the hybrid solver performs well and in some cases better than other state-of-the-art classical heuristics on comparable hardware<sup>15</sup>. Additionally, when running the hybrid solver with the quantum annealing augmentation disabled, the solution time and quality are degraded slightly on average. This indicates that the quantum annealing solutions are providing improved performance, at least for the problem types and instances studied to date. The long-term expectation is that as the quantum annealing hardware improves in the future, this should provide even more advantage over the purely classical solvers alone. Interestingly, this approach can also be pursued using other quantum computing paradigms, such as circuit model discrete optimization solvers. Further testing will need to be done on future quantum hardware and quantum algorithms—both quantum annealing and circuit model—to see how much further these hybrid approaches can be developed.



**Figure 1.** A functional diagram of the D-Wave Systems hybrid solver. The Metasolver on the left governs the overarching algorithm. It launches multiple threads which use classical heuristic algorithms running on CPUs or GPUs as depicted in the middle of the figure. Simultaneously, it uses the quantum annealer (QA) on the right to search for promising solutions of smaller subsets of the problem to provide additional promising starting points to the heuristic algorithms.

## Single truck PUBO

The enormously complex logistical routing problem (LRP) described in “[Aisin Corporation vehicle routing problem](#)” could be formulated as the problem of minimizing a polynomial with binary variables. However, such a formulation would involve an intractably large number of variables and may have very high polynomial degree. This complexity suggests that we find another approach, even if that approach cannot achieve an exactly optimal solution.

Consider a single truck and ignore completely the presence of all other trucks. We do not concern ourselves with the fact that other trucks can potentially interfere with or assist this one truck. We only focus on one single truck and we write down a PUBO, the solution to which ought to give a reasonable route for this truck.

**Binary variables.** Suppose that there are  $n$  nodes  $z_0, \dots, z_{n-1}$ . (Note that we are starting indices at 0 for convenience here.  $z_0$  is not a special node as in “A. Basic vehicle routing problem”) The truck starts at one of the nodes at the first time step  $t = 0$  and then drives to another one at each subsequent time step. Assume that there are a total of  $\tau$  time steps so that the route of the truck is  $\xi_0, \xi_1, \dots, \xi_{\tau-1}$  with each  $\xi_t \in \{0, 1, \dots, n - 1\}$ . The fact that we are fixing the total number of time steps will play an important role below.

We now introduce  $n \tau$  Boolean variables

$$\{x_{it} \mid i \in \{0, 1, \dots, n - 1\}, t \in \{0, 1, \dots, \tau - 1\}\}.$$

The intended meaning of these variables is that  $x_{it}$  is supposed to be 1 if the truck is located at node  $i$  at time step  $t$  and  $x_{it} = 0$  otherwise. This approach to introducing binary variables is commonplace when dealing with PUBO formulations of routing problems like the traveling salesman problem.

**Locality term.** Configurations of the binary variables  $x_{it}$  can violate locality: if  $x_{25} = x_{35} = 1$ , then the truck is apparently at  $z_2$  and  $z_3$  simultaneously at time step 5. For this reason, the PUBO will require terms that act as constraints to enforce locality. We must demand that

$$\sum_i x_{it} = 1$$

for all times  $t$ . There are a few ways to handle this, but our approach is to start our PUBO with a quadratic term of the form

$$f_{\text{local}}(x) = A_{\text{local}} \sum_t \left( 1 - \sum_i x_{it} \right)^2 \quad (4)$$

which is 0 if and only if the configuration is local at all times. The coefficient  $A_{\text{local}}$  is some positive real number which can be tuned to improve an algorithm’s performance.  $f_{\text{local}}(x)$  is positive when the configuration is nonlocal and it gets larger as the solution becomes more and more non-local.  $f_{\text{local}}$  also prohibits configurations where all variables are zero at any given time.

**Demand and time.** The tensor demand structure of the LRP explained in “[Tensor demand structure](#)” poses a challenge for a PUBO formulation. Off-board demand changes as trucks pick up and drop off boxes, and we thus need a dynamical approach to deal with demand to be technically correct. However, the single truck PUBO we are building is a heuristic, so we allow a heuristic approach at this point.

Consider rank-2 and rank-3 off-board demand  $D_{ij}^{r=2}$  and  $D_{ijk}^{r=3}$ . (To review, the rank-3 demand means material which starts at node  $i$  and must be brought to node  $j$  and then to node  $k$ .) We can use this demand to compute an “overall off-board demand”:

$$\bar{D}_{ij} = D_{ij}^{r=2} + \sum_k \left( D_{ijk}^{r=3} + D_{kij}^{r=3} \right). \quad (5)$$

This overall demand can be interpreted as an upper bound on the amount of material that a truck could encounter at node  $i$  that needs to be sent *directly* to node  $j$ . (Note that we exclude from this accounting rank-3 demand in the summation that would be sent *indirectly* from node  $i$  to node  $j$  via an intermediate node  $k$ , as those cases will be handled by a sequence of two steps using two other terms in the objective function summation described immediately below.)

We then introduce a demand term for the PUBO:

$$f_{\text{demand}}(x) = -A_{\text{demand}} \sum_{ijt} \sum_{\delta=1}^{\delta_*(t)} \bar{D}_{ij} x_{it} x_{j,t+\delta} \quad (6)$$

where  $\delta_*(t) = \min(\delta_{\max}, \tau - 1 - t)$  and  $\delta_{\max}$  is some fixed integer limitation on  $\delta$ . The reason that we include  $\delta > 1$  terms is that a truck may pick something up from node  $i$  with destination at node  $j$ , but the truck might stop at node  $k$  before going to node  $j$ . In this case, we still want to reward the truck from driving from  $i$  to  $j$  eventually. Solutions which approximately minimize  $f_{\text{demand}}$  may correspond to routes where the truck can carry large amounts of demand.

We can similarly penalize routes that take excessively long drives. If  $T_{ij}$  is the time matrix computing the driving time between nodes, adding

$$f_{\text{time}}(x) = A_{\text{time}} \sum_{ijt} T_{ij} x_{it} x_{jt+1} \quad (7)$$

to the PUBO will add a preference for routes that can visit more nodes in the limited number of time steps. Note that terms (6) and (7) can be combined into one quadratic term with modified coefficients.

**Redundancy abatement.** The final term to build our PUBO requires some explanation. Consider again the “overall demand” matrix  $\bar{D}_{ij}$ . There are two situations to consider: when  $\bar{D}_{ij}$  is large and when it is small. When it is large, our usage of  $\bar{D}_{ij}$  is quite reasonable. Configurations that try to minimize  $f_{\text{demand}}$  and  $f_{\text{time}}$  are likely to satisfy lots of demand while avoiding long times. However, in the case where  $\bar{D}_{ij}$  is small we encounter a serious deficiency. As mentioned above, off-board demand, when treated without any simplification, is dynamical: it changes as trucks pick up and drop off parts.

Consider the following extreme example. Suppose that between nodes  $z_3$  and  $z_6$ , there is a small amount of demand  $\bar{D}_{36}$ . If our truck drives from  $z_3$  to  $z_6$  once, doing so a second time would be pointless. However, there is no information that this would be wasteful in  $f_{\text{demand}}$ . In fact,  $f_{\text{demand}}$  may suggest that going back and forth between  $z_3$  and  $z_6$  for all time steps is a good solution! To avoid this sort of thing, we introduce terms in the PUBO which penalize route repetition. However, we only want to do this for pairs of nodes where  $\bar{D}_{ij}$  is “small” so we will only introduce these new terms for certain pairs of nodes. We will use the notation  $I_{ij} = 1$  when  $\bar{D}_{ij}$  is sufficiently “small” that we would want to include such a term and we will write  $I_{ij} = 0$  for “large”  $\bar{D}_{ij}$ .

In the example above, including a term proportional to

$$\sum_{t=0}^{\tau-4} x_{3t} x_{6t+1} x_{3t+2} x_{6t+3}$$

counts the number of times that a truck follows the path  $z_3 \rightarrow z_6 \rightarrow z_3 \rightarrow z_6$ . We can generalize this term further by adding a delay  $\delta \in \{2, \dots\}$  between repetitions as follows:

$$h_\delta(3, 6) = \sum_{t=0}^{\tau-2-\delta} x_{3t} x_{6t+1} x_{3t+\delta} x_{6t+1+\delta}$$

which is the same as before except now we are counting the number of times that the truck follows the two-step path  $z_3 \rightarrow z_6$  at some time  $t$  and then later follows the same path at time  $t + \delta$ . We can then define

$$h(i, j) = \sum_{\delta=2}^{\tau-2} \sum_{t=0}^{\tau-2-\delta} x_{it} x_{jt+1} x_{it+\delta} x_{jt+1+\delta}$$

Finally, we can give the redundancy abatement term for our PUBO

$$f_{\text{nonredundant}}^I(x) = A_{\text{nonredundant}} \sum_{ij} I_{ij} h(i, j) \quad (8)$$

where  $I$  is an  $n \times n$  matrix consisting of only zeros and ones which is supposed to be 1 for pairs  $ij$  if and only if demand  $\bar{D}_{ij}$  is small enough to include such a repetition-avoidant term. Note that equation (8) is fourth-order, and we should thus use these terms sparingly:  $I_{ij}$  should be zero unless we expect cycling between  $i$  and  $j$  to be a serious problem.

**Full single truck PUBO.** Combining the ideas from above, our full single-truck PUBO is the summation of Eqs. (4), (6), (7), and (8). There are four positive constants that can be tuned:  $A_{\text{local}}$ ,  $A_{\text{demand}}$ ,  $A_{\text{time}}$ , and  $A_{\text{nonredundant}}$  as well as the choice of  $I_{ij}$  and  $\delta_{\max}$ . These parameters offer substantial freedom to tune heuristic PUBO-solvers to obtain better solutions for the final combined problem instance.

**Solution rectification.** Approximate solutions to a single-truck PUBO do not always give a well-defined route for a truck. It’s possible that a solver returns a solution which is not a minimum of  $f_{\text{local}}$ . For these situations, it’s helpful to settle on a method for “rectifying” solutions that violate locality.

To enforce locality on an approximate solution  $x$ , we iterate through time steps  $t$ . If we have  $x_{it} = 0$  for all  $i$ , then we randomly choose one node  $i_*$  to have  $x_{i_*t} = 1$  while keeping  $x_{it} = 0$  when  $i \neq i_*$ .

If, for some  $t$ , our initial solution has multiple nodes  $i_1, i_2, \dots, i_k$  with  $x_{it} = 1$ , then we wish to select a single value of  $s_* \in \{1, \dots, k\}$  to keep  $x_{i_*t} = 1$  while setting the other variables equal to 0. To do this, we simply look at the terms of the PUBO that involve this time step  $t$  and we try the  $k$  possibilities by brute force. We select any of the choices with the lowest objective function value.

**Order reduction.** To submit a PUBO on D-Wave Systems solvers, including their quantum annealer as well as simulated annealing and the D-Wave hybrid solver, there is a necessity to eliminate any polynomial terms with order exceeding two.

Polynomial order can be reduced at the cost of introducing extra variables and constraints. This is easy to see through an example. Given a sufficiently large real number  $\lambda$ , the polynomial of binary variables  $f(x, y, z) = xyz$  is “equivalent” to  $g(x, y, z, a) = az + \lambda(a - xy)^2$  in the sense that the minimum of  $g$ , restricted to the variables

$(x, y, z)$ , is also a minimum of  $f$ . To see this, note that the variable  $a$ , being binary, takes on the same values as  $xy$  (0 and 1 only). Moreover, the term  $\lambda(a - xy)^2$  enforces the constraint that  $a = xy$  to minimize  $g$  as long as  $\lambda$  is large enough.

In the executions discussed in “[Execution and performance](#)”, we perform PUBO to QUBO conversions for simulated annealing and for D-Wave hybrid runs. For all cases, we used the open-source library Qubovert<sup>29</sup> to perform PUBO to QUBO conversions.

## Supply chain workflow

An approximate solution to the single-truck PUBO described in [Single truck PUBO](#) gives a route for one truck, but the logistical routing problem (LRP) of “[Aisin Corporation vehicle routing problem](#)” involves many trucks. Constructing a PUBO with solutions that obtain multi-truck solutions for a multi-truck VRP is possible, but will involve a large number of variables and terms. We certainly do not consider such an approach to be viable with hardware such as quantum annealers constructed by D-Wave Systems.

The single-truck PUBO, on the other hand, can viably be attacked with near-term quantum algorithms. We thus devise an algorithm which attempts to solve the full LRP by working truck-by-truck, iteratively obtaining a route, estimating the demand satisfied by the route, and then updating the remaining off-board demand tensor before proceeding to the next truck.

For this truck-loop algorithm, we work only with “overall demand” given in Eq. (5). This demand is a useful simplification to avoid the challenges of higher-rank demand while still obtaining a reasonable solution. We return to the details of higher-rank demand in “[B. Full-scale simulation](#)”.

The truck-loop algorithm is given in pseudo code as follows:

---

**Algorithm1** Truck Loop

---

Inputs:

- Number of time steps to use  $\tau$
- Stopping condition `stop`
- Solver with rectification `solver`
- Initial overall demand  $D$
- `routes` ← empty list
- $\ell \leftarrow 0$

```

while not stop( $D, m$ ) do
     $f \leftarrow \text{single\_truck\_pubo}(D, \tau)$ 
     $\xi \leftarrow \text{solver}(f)$ 
     $D \leftarrow D - \text{estimate\_demand}(D, \xi)$ 
    routes ← append(routes,  $\xi$ )
     $\ell \leftarrow \text{length of routes}$ 
end while
return routes

```

---

This algorithm refers to some subroutines that require explanation:

- `single_truck_pubo` returns a PUBO as described in “[Single truck PUBO](#)”. The only part of this function that cannot be read-off from [Single truck PUBO](#) is the choice of redundancy abatement coefficients  $I_{ij}$ . These are chosen to be 0 for a pair  $i, j$  when  $D_{ij}$  is above some threshold and 1 otherwise.
- `solver` refers to some PUBO solver and post-processor. The PUBO solver can be any exact or heuristic algorithm which is aims to find the minimum of a given PUBO. The output of `solver` is not, however, the min or argmin of the PUBO. Instead, we first get the binary variable assignments that approximately minimize the PUBO, and then we perform locality rectification as explained in “[Single truck PUBO](#)”.

Moreover, if there are driving windows (see “[Restricted driving windows](#)”), then we also cut off the last steps of the route until the route fits into the window. Along the same lines, we may also lengthen routes that fall well-short of the driving window. This can be done by looping the original route until adding one more step would exceed the time window. This procedure may sound counterproductive, but in applications with driving windows, as in the ALRP (“[Aisin Corporation vehicle routing problem](#)”), we may calculate cost by assuming that all trucks have a fixed price for a fixed window, and in that case it’s ideal to use the truck throughout the time window.

- Finally, the output of `solver` is the sequence of nodes for the single-truck route obtained.
- `estimate_demand` is a function which attempts to estimate the degree to which  $D$  will be reduced by a truck following a given route. We compute this by performing a small supply chain simulation for the route. Suppose that the truck starts at node  $z_4$  and that it will then go to nodes  $z_6, z_8, z_3$  in that order. We then load onto an abstract truck the largest allowed amount of  $D_{4,6}$  (limited either by the truck capacity or by the value of  $D_{4,6}$ ). If there is remaining truck capacity we move on to  $D_{4,8}$ , and so on until we either take all of  $D_{4,3}$  or we run out of capacity. At this point, the truck (abstractly) drives to  $z_6$  and we unload all of the  $D_{4,6}$  that was on-board. We then repeat the procedure starting with  $D_{6,8}$  followed by  $D_{6,3}$ . This method provides a reasonable estimate of the demand that a truck is expected to satisfy, but it is only a heuristic approach, severely weakened by the realities of multi-truck interactions.
  - `stop` is a stopping condition such that `stop`( $D, m$ ) is true if and only if conditions on a demand matrix  $D$  and the number of trucks assigned  $m$  are satisfied. We can, for example, stop whenever more than 50 trucks have been assigned or when all entries of  $D$  are below some threshold.

Once the truck loop algorithm is complete, we obtain a collection of routes for each truck

$$\xi = \{\xi_m \mid m = 1, \dots, N\} \quad (9)$$

where  $N$  is the number of trucks and each  $\xi_m$  is a list of nodes  $(\xi_{m,1}, \xi_{m,2}, \dots, \xi_{m,k_m})$  with  $k_m \leq \tau$  for all  $m$ . Unfortunately, this final routing was obtained with a series of heuristics and it's therefore not immediately obvious how to estimate the solution quality. The most naive and easily implemented strategy is to simply look at the final value of  $D$  in the algorithm and the total time for all of the trucks to drive along the routes. However, this demand and time calculation is only a approximation. To determine the quality of the solution, a more involved supply chain simulation is necessary, and we explain this in “[Full-scale simulation](#)”.

### Execution and performance

The purpose of the truck loop algorithm of “[Supply chain workflow](#)” is to have an algorithm that can break a very large commercial problem into sufficiently small pieces that simulated and quantum annealing algorithms can viably be applied in the near term. As a proof-of-concept, we applied our algorithm to proprietary supply chain data of Aisin Corporation using both simulated annealing and the D-Wave Hybrid algorithm. Application of direct quantum annealing rather than the D-Wave Hybrid algorithm remains unrealistic for this problem until the hardware matures to some extent.

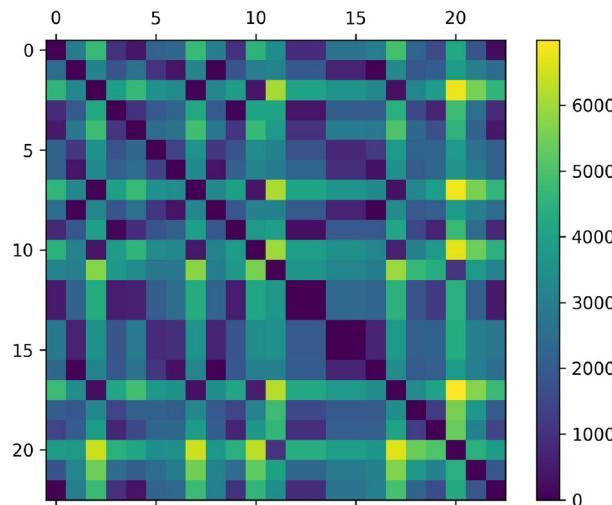
We referred to the Aisin problem, which was referred to as the ALRP in “[Aisin Corporation vehicle routing problem](#)”. Their data consists of approximately 350,000 boxes to be shipped by trucks among 23 nodes in Japan. The travel times between nodes is shown in Fig. 2. The boxes have rank-2 and rank-3 demand structure. Thus, some boxes have direct delivery requirements and others have a required stop along the way. To apply the truck loop algorithm, we first group boxes by their required routes and then we apply the “[box soup](#)” simplification of “[Individual boxes and the box soupsimplification](#)” to convert from a discrete to continuous demand structure. Moreover, we use Eq. (5) to compute the overall demand matrix. With truck volumes normalized to 1, the overall demand matrix is illustrated in Fig. 3.

**Truck loop annealing runs.** We ran the truck loop algorithm of “[Supply chain workflow](#)” using both a simulated annealing algorithm (“B. 1. Simulated annealing”) and the D-Wave Hybrid solver (“B. 2. Quantum annealing”).

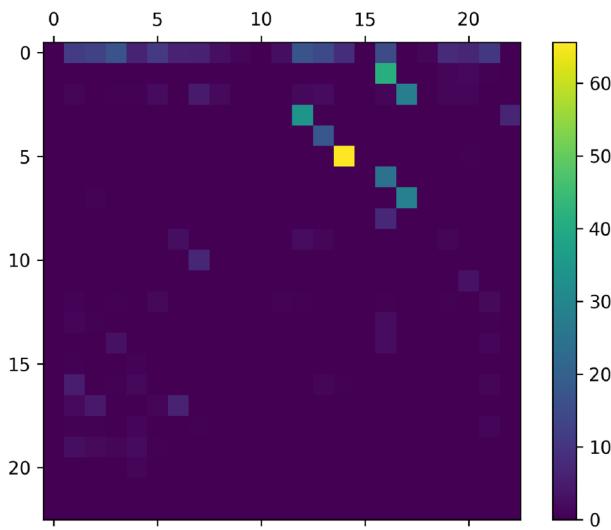
During each step of the loop, we computed an estimated overall demand carried by each truck. That estimated overall demand is shown for the case of simulated annealing and D-Wave Hybrid in Figs. 4 and 5, respectively. The simulated annealing loop terminated after 61 trucks, and the D-Wave Hybrid loop terminated after 74 trucks. This termination condition, defining `stop` from “[Supply chain workflow](#)”, is simply to stop all entries of  $D$  under a very small cutoff (.0005 in our case).

To construct the single-truck PUBO, we used a different value of  $\tau$  (the number of time steps) for simulated annealing and for the D-Wave Hybrid solver: 15 for simulated annealing and 5 for D-Wave Hybrid. With 15 time steps, the typical number of PUBO variables is approximately 350, corresponding to approximately 2500 QUBO variables. With only 5 time steps, there are around 100 PUBO variables which are equivalent to about 200 QUBO variables.

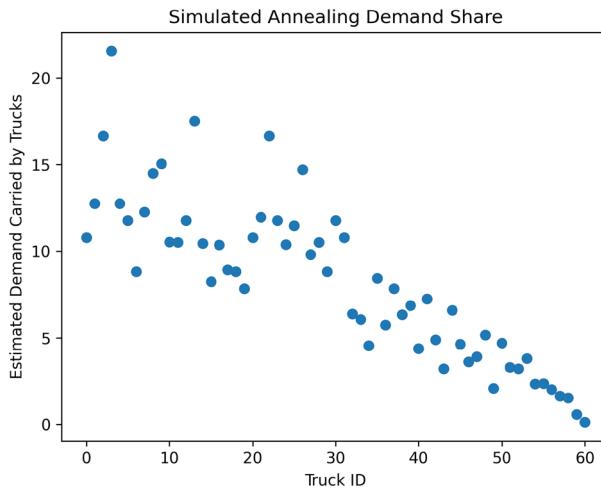
The PUBO coefficients were the same in both cases:



**Figure 2.** The matrix of driving times between the 23 nodes in the Aisin logistical routing problem. Units are in seconds which is the same unit used for the PUBO construction (figure generated by the authors using matplotlib 3.6.3 found at <https://pypi.org/project/matplotlib/>).



**Figure 3.** The overall demand matrix (as defined in Eq. (5) for the Aisin Logistics Problem of “[Aisin Corporation vehicle routing problem](#)”. Note that of the  $23^2 = 529$  entries for the matrix, only 115 are nonzero (figure generated by the authors using matplotlib 3.6.3 found at <https://pypi.org/project/matplotlib/>).

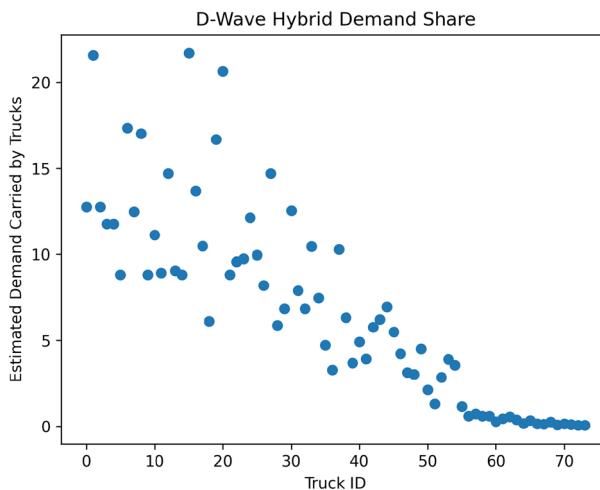


**Figure 4.** Estimated demands computed during the truck loop process for simulated annealing. The algorithm terminated after 61 trucks were given routes. Demands are estimated by the method explained in “[Supply chain workflow](#)”.

$$\begin{aligned} A_{\text{local}} &= 5000, \\ A_{\text{demand}} &= 320, \\ A_{\text{time}} &= .01, \\ A_{\text{nonredundant}} &= 1. \end{aligned}$$

As we emphasized earlier, the data in Figs. 4 and 5 cannot, on their own, be taken as an evaluation of the performance of our methodology. This is especially true given that demands are only estimated by the methods explained in “[Supply chain workflow](#)”. We now explain how we translated the routes and heuristic solutions from this section into meaningful instructions for a commercial supply chain and, in doing so, evaluate the performance of our workflow.

**Full-scale simulation.** The truck loop algorithm provides a simple way to break up the enormous logistical routing problem (“[Aisin Corporation vehicle routing problem](#)”) into smaller problems relating to one truck at a time. The output of the truck loop algorithm is a set of routes  $\xi$  as in Eq. (9). These routes were computed with methods that ignored most of the inter-truck interaction and also treated boxes as a continuous “box soup”. How can we apply the routes  $\xi$  in a more realistic fashion given those simplifications?



**Figure 5.** Estimated demands computed during the truck loop process using the D-Wave Hybrid solver. The algorithm terminated after 74 trucks were given routes. Note that the last 15 trucks consume a very small amount of demand, suggesting that they can be replaced by a smaller number of trucks. Demands are estimated by the method explained in “Supply chain workflow”.

We chose to interpret the output  $\xi$  as “suggested routes” and to attempt to use them in a *full-scale* supply chain simulation. The simulation we built does not make any of the simplifications described in “Single truck PUBO” or “Supply chain workflow”. Boxes are treated as real boxes with identity. Every box is individually tracked, has a specified volume, and every box has given rank-2 or rank-3 requirements. In other words, boxes have required paths as in Eq. (2), but those paths have either length 2 or length 3. All of this detail is in the supply chain data set of Aisin Corp.

While we have dropped the simplifying assumptions, the simulation is not exact and global in that we are starting with the fixed truck routes determined by our quantum-classical heuristic in the truck loop algorithm. Trucks follow along the routes they were assigned by the truck loop algorithm and they pick up as many boxes as is sensible for their route. The method of pickup selection is very similar to that described for `estimate_demand` in “Supply chain workflow”. We briefly describe it here for clarity. Suppose that a truck follows the route  $z_1, z_2, z_3, z_4$ . When the truck starts (at  $z_1$ ), it first tries to pick up demand destined for  $z_2$ . We thus look at all boxes currently at  $z_1$  with their next stop being  $z_2$ . As many such boxes (with no bias for which ones) are picked up as can fit on the truck. (Note that here we continue to use the simplification that box volumes are fluid; we do not compute close packing of the three-dimensional shape of boxes.) Then we ask if there is space for boxes at  $z_1$  that are destined for  $z_3$  that still fit on the truck. The same is done for  $z_4$  if there is remaining space. At this point, the truck drives to  $z_2$ .

During this time, other trucks will be driving and will alter the material at various nodes. We simulate in detail what each truck is doing at this time step and modify the demand as appropriate. Then we advance to the next time step. When our example truck arrives at  $z_2$ , we first drop off all boxes that need to go to  $z_2$  as their next stop. We then repeat the prior procedure, asking if there are boxes at  $z_2$  that need to go to  $z_3$  and so on. We repeat this for each truck in this time step and then advance to the next time step and repeat. Our simulation also takes into account exact details of driving windows, ensuring that no driving goes outside of these windows. The simulation runs until the end of the set maximum time—there is no other stopping condition.

This detailed simulation converts a list of suggested routes for trucks into a precise history of what every truck in a supply chain does and the path of every box in the supply chain. Such simulation is an important tool because it allows algorithms like our truck loop algorithm to make various simplifications while still getting a final result that is commercially useful.

One more utility of the exact simulation, is that it allows for making final corrections to routes and knowing for a fact that those corrections improve performance. In the description above, the trucks routes returned from the truck loop algorithm were held fixed for the duration of the simulation, but there are many ways one could consider variations of the routes that might improve performance. For our purposes, we used a very simple heuristic to modify routes: after running a simulation, we checked to see if there were trucks which end their routes on a sequence of steps carrying no demand whatsoever. We clipped these parts of the routes away, and replaced them with a route driving back and forth between whatever two nodes had the highest unsatisfied demand during that simulation run. We then performed the simulation again, and kept it if the overall satisfied demand was superior. We performed this step only 5 times for each overall run, and found only modest improvements.

The simulation algorithm, including the five-variation repeat, is given in pseudocode as follows:

**Algorithm2** Simulation

---

Inputs:

- Number of variation loops  $m$
- Maximum drive time  $\tau$
- Number of trucks  $n$
- List of  $n$  truck routes `routes(n)`
- Specification of node locations `nodes`
- Parts list (ID, Volume V, Overall Demand  $D$ )
- `best_routes(n) ← empty list`

```

for variation=1 to  $m$  do
    new_best_routes(n) ← empty list
    for time=0 to  $\tau$  do
        for truck=1 to  $n$  do
            compute location of truck at time
            if truck at node then
                 $D \leftarrow$  update pick-up/drop-off demand
                new_best_routes(truck) ← update delivery schedule for truck
            end if
        end for
    end for
    if new_best_routes(n) better than best_routes(n) then
        best_routes(n) ← new_best_routes(n)
    end if
    if best_routes(n) have later stops with no pick-up/drop-off then
        routes(n) ← Modify routes by redirecting to unmet demand
    end if
    end for
return best_routes(n)

```

---

For both the simulated annealing routes and the D-Wave Hybrid routes, we kept only the first 61 trucks. We found that these trucks satisfied 96.45% of the overall Aisin Corp. demand when using simulated annealing and 99.39% with the D-Wave Hybrid solver. We make no claim that these results would compete with dedicated classical optimization tools commonly used in operations research. However, these results illustrate the viability of breaking problems into small pieces, running those pieces on solvers that can only run smaller problem sizes (like near-term quantum algorithms and quantum annealers), and restoring a realistic solution through simulation.

The truck routing found through our methods using the D-Wave hybrid solver followed by the full-scale simulation with 61 trucks is depicted in Fig. 7 whereas the routes devised by Aisin Corporation logistics experts are shown in Fig. 6. The truck routes that we found have a substantially greater degree of connectivity than those currently used in practice.

**Comparison with current supply chain performance.** Aisin Corporation currently routes trucks in their supply chain by careful analysis by logistics experts. In this subsection, we briefly compare the current routes used by Aisin Corporation with the routes determined by the execution of our workflow. A major finding is that our routes appear to outperform current routing by a number of metrics.

The connectivity graph currently used by Aisin Corporation (that is, the pairs of nodes that trucks currently drive between) is shown in Fig. 6. Our execution on the D-Wave hybrid solver, followed by full-scale simulation with 61 trucks as described in “Execution and performance” and “Full-scale simulation”, yielded the substantially denser connectivity graph shown in Fig. 7. The routes currently used involve 142 trucks whereas our methods use only 61 trucks, implying a large reduction in cost.

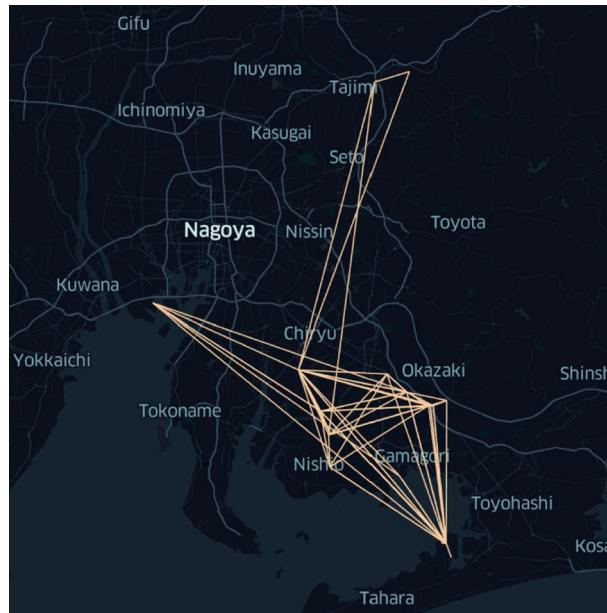
These are exciting results, but we caution that our simulations are not complete in all regards. There may be constraints that logistics experts must account for that we have not been able to include in the simulation of “Full-scale simulation”, such as how long a parts box needs to stay at an intermediate node (we assume immediate availability) or how the drive time varies during the day due to traffic conditions, so the overall reduction in truck number may not be as dramatic as it appears. Nonetheless, using our routes as a starting point and making small adjustments is expected to yield a meaningful cost reduction.

A less obvious benefit of the workflow we describe is that it yields very precise instructions for each truck’s route and delivery requirements. An example of such a routing is shown in Fig. 8 (this figure only shows the requirements for two parts, referred to as part A and part B). Because there are approximately 15,000 parts, such a precise prescription is extremely useful. Aisin Corporation currently relies on human experience and intuition to make precise determinations for part pickup requirements, while our approach removes this pain point. We emphasize, however, that specific pickup and dropoff instructions are determined by the full-scale simulation implementation, and are not directly determined by the PUBO optimizer.

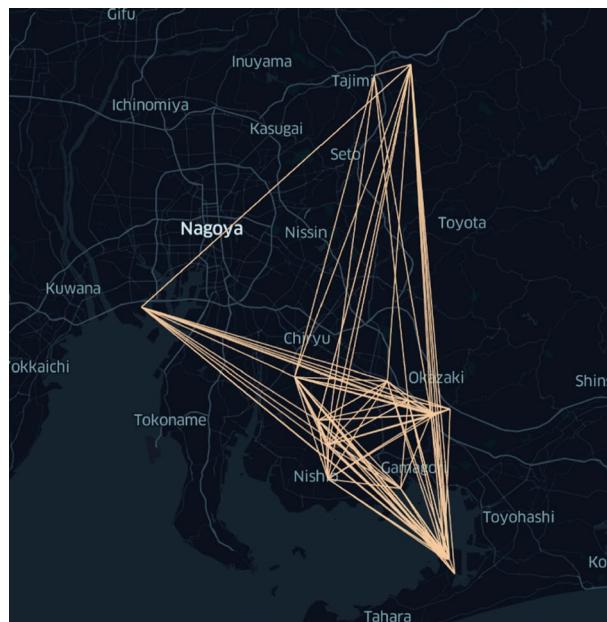
## Conclusions

Quantum algorithms are known to provide computational benefits over classical computing for specific tasks. This specificity implies that, even when quantum hardware matures, quantum computing will most likely be used as a component in hybrid classical-quantum workflows when dealing with complex problems of practical significance.

The methodology of this work is an example of such a hybrid workflow. Even for the enormously complex task of routing trucks in a realistic supply chain, we were able to find a way to construct small problem instances



**Figure 6.** Connectivity graph for routes currently used by Aisin Corporation. Lines indicate pairs of the 23 nodes that trucks drive between in the routes determined by Aisin Corporation logistics experts. For this routing, 142 trucks deliver approximately 340,000 boxes containing  $\sim$  15,000 unique parts. Among these parts, there are 115 unique routing requirements as described in “Individual boxes and the box soupsimplification” (figure generated by the authors using kepler.gl v3.0.0-alpha.0 found at <https://kepler.gl/>).



**Figure 7.** Connectivity graph for routes found through our workflow with 61 trucks, the D-Wave hybrid solver, and full-scale simulation (see “Full-scale simulation”). The problem instance is the same as described in the caption of Fig. 6 (figure generated by the authors using kepler.gl v3.0.0-alpha.0 found at <https://kepler.gl/>).

Time	Place	Part A Load	Parts A Unload	Part B Load	Parts B Unload
2021/2/25 9:00	Logistic Center A	20	0	1	0
2021/2/25 10:18	Logistic Center B	0	20	0	1
2021/2/25 10:36	Warehouse C	0	0	0	0
2021/2/25 10:59	Factory D	0	0	0	0
2021/2/25 13:35	Logistic Center A	0	0	0	0
2021/2/25 15:10	Logistic Center B	2	0	0	0
2021/2/25 15:28	Warehouse C	0	0	0	0
2021/2/25 15:51	Factory D	0	2	0	0
2021/2/25 17:27	Logistic Center A	18	0	0	0
2021/2/25 22:03	Logistic Center B	13	18	0	0
2021/2/25 22:21	Warehouse C	0	0	0	0
2021/2/25 22:44	Factory D	0	13	0	0
2021/2/26 1:20	Logistic Center A	17	0	0	0
2021/2/26 2:55	Logistic Center B	18	17	0	0
2021/2/26 3:13	Warehouse C	0	0	0	0
2021/2/26 3:36	Factory D	0	18	0	0
2021/2/26 5:12	Logistic Center A	0	0	0	0

**Figure 8.** Example of a detailed route found for a truck after D-Wave hybrid execution followed by full-scale simulation. This table only includes loading and unloading instructions for two of approximately 15,000 parts, all of which are accounted for in our workflow.

as subroutines that can be solved with NISQ quantum algorithms. For our study of a commercial supply chain, the smaller problems are binary optimization problems with  $\sim 2500$  binary variables, an appropriate scale for near-term quantum annealing and somewhat near-term circuit model quantum computing. When running our workflow with solvers like simulated annealing and the D-Wave hybrid solver, proxies used in lieu of more mature quantum hardware, we found viable solutions for the full supply chain.

We do not claim that our method furnishes a provable performance advantage over classical algorithms because NISQ quantum optimization techniques like the quantum annealing and QAOA are heuristics without relevant proven guarantees. However, the individual truck routing binary optimization problem is itself NP-hard, and can thus be a performance bottleneck. It's sensible to apply quantum algorithms to these bottlenecks, and this approach will pay off if NISQ quantum optimization algorithms that outperform classical computing are discovered.

Beyond vehicle routing, we believe that our approach is viable for a wide range of practically important optimization problems. Many problems can be approximately solved through heuristics that decompose a large problem into solving a large number of smaller problems, and such smaller problems can often be reduced to binary optimization. As long as the number of variables for those small problems is appropriate for NISQ quantum algorithms, such a hybrid approach can be used as a way to explore the performance of quantum optimization algorithms in the use case without resorting to miniaturizing the actual problem.

## Data availability

The data sets and code generated and/or analysed during the current study are not publicly available due to proprietary commercial restrictions. This specifically applies to the details of the logistics routes and parts delivery demand structure. However, the formulation of the algorithm is fully described herein so that interested parties could generate their own logistics scenarios with depot/factory locations and supply pick-up/delivery demand structure and apply the algorithm.

Received: 29 November 2022; Accepted: 16 March 2023

Published online: 23 March 2023

## References

- Shor, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.* **41**(2), 303–332 (1999).
- Feynman, R.P. Simulating physics with computers. in *Feynman and Computation*. 133–153 (CRC Press, 2018).
- Lloyd, S. Universal quantum simulators. *Science* **273**, 1073–1078 (1996).
- Arute, F. *et al.* Quantum supremacy using a programmable superconducting processor. *Nature* **574**(7779), 505–510 (2019).
- Farhi, E., Goldstone, J. & Gutmann, S. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028* (2014).
- Kadowaki, Tadashi & Nishimori, Hidetoshi. Quantum annealing in the transverse Ising model. *Phys. Rev. E* **58**(5), 5355 (1998).
- Farhi, E., Goldstone, J., Gutmann, S. & Sipser, M. Quantum computation by adiabatic evolution. *arXiv preprint arXiv:quant-ph/0001106* (2000).
- Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5(1):1–7, 2014.
- Preskill, J. Quantum computing in the NISQ era and beyond. *Quantum* **2**, 79 (2018).
- Johnson, M. W. *et al.* Quantum annealing with manufactured spins. *Nature* **473**(7346), 194–198 (2011).
- Ray, P., Chakrabarti, B. K. & Chakrabarti, A. Sherrington-Kirkpatrick model in a transverse field: Absence of replica symmetry breaking due to quantum fluctuations. *Phys. Rev. B* **39**, 11828–11832. <https://doi.org/10.1103/PhysRevB.39.11828> (1989).
- Morita, S. & Nishimori, H. Convergence theorems for quantum annealing. *J. Phys. A Math. General* **39**(45), 13903. <https://doi.org/10.1088/0305-4470/39/45/004> (2006).

13. Das, A. & Chakrabarti, B. K. Colloquium: Quantum annealing and analog quantum computation. *Rev. Mod. Phys.* **80**, 1061 (2008).
14. Albash, T. & Lidar, D. A. Adiabatic quantum computation. *Rev. Mod. Phys.* **90**, 015002 (2018).
15. King, J. *et al.* Quantum annealing amid local ruggedness and global frustration. *J. Phys. Soc. Jpn.* **88**(6), 061007. <https://doi.org/10.7566/JPSJ.88.061007> (2019).
16. McGeoch, C., Farre, P. & Bernoudy, W. *Technical Report: D-Wave Hybrid Solver Service + Advantage: Technology Update*. [https://www.dwavesys.com/media/m2xbmlhs/14-1048a-a\\_d-wave\\_hybrid\\_solver\\_service\\_plus\\_advantage\\_technology\\_update.pdf](https://www.dwavesys.com/media/m2xbmlhs/14-1048a-a_d-wave_hybrid_solver_service_plus_advantage_technology_update.pdf) (2020).
17. *White Paper: Hybrid Solvers for Quadratic Optimization*. <https://www.dwavesys.com/media/soxph512/hybrid-solvers-for-quadratic-optimization.pdf> (2022).
18. Dantzig, G. B. & Ramser, J. H. The truck dispatching problem. *Manag. Sci.* **6**(1), 80–91 (1959).
19. Toth, P. & Vigo, D. *The Vehicle Routing Problem*. (SIAM, 2002).
20. Karp, R.M. Reducibility among combinatorial problems. in *Complexity of Computer Computations: Proceedings of a Symposium on the Complexity of Computer Computations*. 85–103. [https://doi.org/10.1007/978-1-4684-2001-2\\_9](https://doi.org/10.1007/978-1-4684-2001-2_9) (1972).
21. Lucas, A. Ising formulations of many NP problems. *Front. Phys.* **2**, 5 (2014).
22. Pincus, M. Letter to the editor—A Monte Carlo method for the approximate solution of certain types of constrained optimization problems. *Oper. Res.* **18**(6), 1225–1228 (1970).
23. Van Laarhoven, P.J.M. & Aarts, E.H.L. *Simulated Annealing: Theory and Applications*. (Reidel Publishing Company, 1987).
24. Aharonov, D., van Dam, W., Kempe, J., Landau, Z., Lloyd, S. & Regev, O. Adiabatic Quantum Computation is Equivalent to Standard Quantum Computation. 42–51. <https://doi.org/10.1109/FOCS.2004.8> (2004).
25. Santoro, G. E. & Tosatti, E. Optimization using quantum mechanics: Quantum annealing through adiabatic evolution. *J. Phys. A Math. General* **39**(36), R393 (2006).
26. Denchev, V. S. *et al.* What is the computational value of finite-range tunneling?. *Phys. Rev. X* **6**, 031015. <https://doi.org/10.1103/PhysRevX.6.031015> (2016).
27. McGeoch, C. & Farré, P. Advantage processing overview, a technical report. in *White Paper*. [https://www.dwavesys.com/media/3xvdipcn/14-1048a-a\\_advantage\\_processor\\_overview.pdf](https://www.dwavesys.com/media/3xvdipcn/14-1048a-a_advantage_processor_overview.pdf) (2017).
28. Adame, J.I. & McMahon, P.L. Inhomogeneous Driving in Quantum Annealers Can Result in Orders-of-Magnitude Improvements in Performance (2018).
29. Iosue, J.T. *Qubovert*. URL v1.2.4. <https://pypi.org/project/qubovert/>.

## Author contributions

The methodology of the study was accomplished as a group by S.W., F.S., and T.I. Joint research supervision was accomplished by K.K. and R.C. T.I. and K.K. provided the supply chain logistics demand structure and composed the logistics solution display shown in figure 6. S.W. and F.S. formulated the quantum-classical hybrid algorithm. S.W. performed the computational runs to assess performance and improve the algorithm. S.W. and R.C. wrote the main manuscript.

## Competing interests

The authors disclose the following financial interests: S.W., F.S., and R.C. were all employed by QC Ware Corp. during their scientific and writing contributions to this manuscript. R.C. is a co-founder of QC Ware Corp. S.W., F.S., and R.C. hold QC Ware stock or stock options. T.I. and K.K. were employed by Aisin Group throughout their respective scientific and writing contributions. Aisin Group is a customer of QC Ware; Aisin Group provided funding for the research presented in this manuscript.

## Additional information

**Correspondence** and requests for materials should be addressed to R.C.

**Reprints and permissions information** is available at [www.nature.com/reprints](http://www.nature.com/reprints).

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

© The Author(s) 2023