Improved Classical and Quantum Algorithms for the Shipment Rerouting Problems

Abstract

In this paper, we study a shipment rerouting problem, which generalizes many NP-hard routing problems and packing problems. This problem has ample and practical applications in vehicle scheduling and transportation logistics. Given a network of hubs, a set of shipments needs must be delivered from their sources to their respective destinations by trucks. The objective is to select a set of transportation means and schedule travel routes so that the total cost is minimized. This problem is NP-hard and only classical approximation algorithms were known to have been studied for some of its NP-hard variants. In [21], a quantum algorithm, based on the Ising model, generates an exact solution for a variant of this problem. In this work, we design classical exact and approximation algorithms as well as a quantum algorithm for this problem. The algorithms that we design use novel mathematical programming formulations and/or new insights on solving packing and routing problems simultaneously. Such algorithms take advantage of the network infrastructure, the shipments, and transportation means. We give provable running time bounds. We conduct extensive experiments and show that our classical solutions are empirically better than the up-to-date quantum algorithms in the noisy intermediate-scale quantum (NISQ) era.

1 Introduction

We consider a generalized version of the shipment rerouting problem studied in [21]. In this problem, there is a network containing hubs and a set of transportation means (such as trucks or trains) that can have their runs on the network. For simplicity, we use trucks with different capacities and different rental/operation costs to represent various transportation means. Some *transportation requests*, which specify some *goods* (also called *needs*) are to be delivered from their sources to their respective destinations, should be satisfied. The objective is to minimize the total cost of successfully transporting the goods from their sources to their destinations.

1.1 Problem statement. Consider the shipment rerouting problem. We have a directed graph $G = (V, E)$ where V is the set of vertices and E is the set of edges. Each edge $e \in E$ has a cost $c_e \geq 0$. A set of goods is to be transported by transportation means (trucks with capacities) running over the network. A subset of vertices $H \subseteq V$ are labeled as *hubs* where goods can be uploaded (at these goods' *source-hubs*) and downloaded (at these goods' *destination-hubs*). Note that a hub can be some goods' source hubs and some other goods' destination hubs.

From the directed graph G, we generate a directed graph $F = (H, P)$ where the set of vertices $H \subseteq V$ denote the hubs only and the set of edges P denote the direct paths connecting two hubs in the transportation network $G = (V, E)$ — that is, the edge $p(i, j) \in P$ in F denotes a shortest path from one hub $i \in H$ to another hub $j \in H$ on the transportation network G and these path-costs $c(p(i, j)) = \sum_{e \in p(i,j)} c_e$ can be pre-calculated using n runs of the Dijkstra's algorithm in time $O(|E||V| + |V|^2 \log |V|)$ where V is the set of vertices in the transportation network G. The generated F is a complete digraph.

Consider the complete digraph $F = (H, P)$. Let $|H| = n$ and we label these hubs $\in H$ as $1, 2, \ldots, n$. There are m *transportation requests* of truck runs for transporting goods. We use a triple (s_i, t_i, l_i) to denote each run, where s_i is the source hub, t_i is the destination hub, and l_i is the goods load. We cannot split these goods during the transportation. There are a set of trucks with various capacities. For goods with load l_i, a truck with its remaining capacity $\geq l_i$ can deliver the goods from s_i to t_i with the travel distance $c(p(s_i, t_i))$. There is a list of K trucks (transportation means) with their capacities as $L_1 \leq L_2 \leq \ldots \leq L_K$. For these trucks, we pay costs $R_1, R_2, \cdots, R_K$ for renting and operating them. Without loss of generality, we have two assumptions: (1) $R_1 \leq R_2 \leq \cdots \leq R_K$ and (2) the functions of rental fee R of load L are concave. We also assume that all the values (edge costs, loads of goods, truck capacities, rental fees) are positive integers. (The model studied in [20] is a special case in which all the trucks are with identical capacity and the same rental cost 0.) The objective is to minimize the total cost of truck rental fees and travel distances to transport all goods from their sources to their respective destinations.

1.2 Related work. Many variants of the shipment rerouting problem have been studied in the past. The general version is intractable, shown as below. Consider a setting in which all the trucks have capacities more than the total goods' load $\sum_i l_i$ and the minimum rental cost R_1 is larger than the total cost of all the paths among all the hubs. In addition, assume that all the transportation needs are very far apart from each other compared to the travel distance of each transportation request. Thus, to minimize the total cost paid to transport all the goods, we only need one truck to ship all the goods, and the optimal solution to our shipment rerouting problem minimizes the total travel distance by visiting all the hubs. The shipment rerouting problem is reduced to a *NP-complete routing problem*, the traveling salesman problem [13]. Now, we consider another special case in which all the transportation needs are with the same source-hub and the same destination-hub. In addition, the network is just a line graph. We show that this variant of the shipment rerouting problem is NP-complete. Consider n transportation requests, each request i having a need l_i. There are only two trucks, with their capacity $= \sum l_i/2$. Thus, we need to decide which request is assigned to which truck to carry from the source to the destination. This problem reduces to an *NP-complete packing problem*, the partition problem [8, 4].

To our knowledge, there are no classical approximation algorithms known for the general shipment rerouting problem. Part of the reason is that an algorithm's approximation ratio is very sensitive to the values in the input instance. It is easy to construct an instance so that an approximation ratio can be changed from 1 to an arbitrary value given a slight change in the shipment loads, the truck capacities, and/or the rental fees. For this problem, we not only consider how to schedule a shipment to minimize the total travel distance but also how to pack goods to satisfy truck capacity constraints. Any single algorithmic technique cannot optimize the objective under both constraints.

In [21], Yarkoni *et. al* studied the version in which all trucks are identical. They formulated the problem as a quadratic unconstrained binary optimization problem and used quantum computers and classic computers to calculate the solution. The formulation used an Ising model-based quantum annealing approach [14]. In designing the logistic network, Ding *et al.* [6] solved the problem using a quantum annealing approach. The constraints are different from the one in [21], and so are the formulations. Neither considered trucks with different capacities and rental fees. Note that allowing trucks to have various capacities makes the packing problem much more challenging.

1.3 Our contributions. In this paper, we design both classical and quantum algorithms for the shipment rerouting problem. From the algorithm design perspective, our contributions include (1) a new integer linear program formulation that generates an exact solution for the general problem, (2) a kernelization technique which pre-processes a given input instance to find out an optimal solution for a subset of instance, (3) an intuitive scalable weighted matching algorithm which optimizes both packing and routing, and (4) an alternative quantum algorithm to the one based on quadratic unconstrained binary optimization (QUBO). From the empirical study perspective, we study our algorithms' performance over various network topologies and transportation requests. The experimental results give us guidance in selecting algorithms for the shipment rerouting problem at various configurations.

In Section 2, we introduce our classical algorithms' techniques and provide their analysis. In Section 3, we introduce our quantum algorithm, which is different from the ones given in [21]. In Section 4, we conduct extensive experiments to compare our algorithmic solutions against the state-of-the-art solutions in the current literature.

2 Classical Algorithms

In this section, we describe our classical algorithms and show their theoretical analysis. We start with some intuition, concepts, and algorithmic insight in Section 2.1. We also include some pre-processing steps using the kernelization techniques in Section 2.1. We then describe two algorithms, one exact solution which is based on a 0-1 integer linear program and one approximation solution which is based on a scalable weighted matching, in Section 2.2 and Section 2.3 respectively.

The algorithm based on the 0-1 integer linear program is named ALG-IP and the algorithm based on the weighted matching is named ALG-WM.

2.1 Preliminaries. Consider an input instance with n hubs on a complete digraph $F = (H, P)$ with costs $c(p(s,t)) \geq 0, \forall s,t \in H, \forall p \in P$. There are m transportation requests with each request i having a source-hub s_i, a destination hub t_i, and a load l_i, where $i = 1, 2, \ldots, m$. There are K trucks with their capacities $L_1, L_2, \ldots, L_K$ and rental/operation fees $R_1, R_2, \ldots, R_K$ respectively. The objective is to minimize the total cost of transporting the goods from their sources to their respective destinations. A solution to this problem is to identify some non-overlapping subsets of requests with each subset being served by a truck under its capacity constraints. An algorithm can be viewed as a process of 'grouping' these m requests so that at the end of the algorithm course, any request belongs to one subset of requests. We use a *group* to represent a set of goods that are to be delivered by a single truck.

Classifying pairs of transportation requests. Let us consider m transportation requests (i.e., (truck-)runs) (s_i, t_i, l_i), where $i = 1, 2, \ldots, m$, with s_i being the source hubs, t_i being the destination hubs, and l_i being the load to be delivered from s_i to t_i. For any pair of two runs, we calculate the increased cost of using one truck for both runs instead of two separate trucks. When two runs are merged, we need to select an appropriate-capacity truck to serve these two runs, based on the order of visiting the sources and the destinations. If two runs are served one after another, we use a truck that has a capacity larger than the heavier-load run. If a truck carries both goods at the same time before dropping anything off, we use a truck whose capacity is larger than the sum of loads. Given two runs (s, t, l) and (s', t', l'), the increased cost (whose value may be negative) due to grouping these two runs into one is defined as (R and R' define the truck rental fees respectively) in Table 1.

case	merged run	increased cost
1	(s, t, s', t')	$c(p(t, s')) - \min(R, R')$
2	(s', t', s, t)	$c(p(t', s)) - \min(R, R')$
3	(s, s', t, t')	$c(p(s, s')) + c(p(s', t)) + c(p(t', t))$ $-c(p(s, t) - c(p(s', t'))$ $+R'' - R - R'$
4	(s, s', t', t)	$c(p(s, s')) + c(p(t', t))$ $-c(p(s, t))$ $+R'' - R - R'$
5	(s', s, t, t')	$c(p(s', s)) + c(p(t, t'))$ $-c(p(s', t'))$ $+R'' - R - R'$
6	(s', s, t', t)	$c(p(s', s)) + c(p(s, t')) + c(p(t', t))$ $-c(p(s, t)) - c(p(s', t'))$ $+R'' - R - R'$

Table 1: The increased cost due to merging two truck runs into one

In Table 1, the value R'' denotes the minimal rental fee for a truck carrying the load $l + l'$. It takes time $O(m^2)$ to calculate the increased cost due to 'merging' every pair of two runs in the requirements. Based on the values calculated above, we label two truck runs as *twisted* or *isolated*, depending on which case (1 to 6 in Table 1) results in the minimum cost increased: If the min-value is one of first two cases (1 and 2), then we name these two truck runs as *isolated* and otherwise (case 3 to case 6), we label them as *twisted.* Such a relationship (twisted or isolated) between two runs is associative but not always transitive. However, our experiments indicate that for most application problems the transitive relationship holds for three or more runs.

Heuristically speaking, a pair of twisted runs usually have close sources and destinations compared to run lengths. A pair of isolated runs are either far apart from each other or have relatively short run lengths. Our classical algorithm (detailed in Section 2.3) is based on this and other intuitions.

Special cases: single source-hub or single destination-hub. Now, we discuss the cases involving shared sources and destinations that can be solved in polynomial time.

LEMMA 2.1. *Consider the case where the transportation network is a star graph with bidirectional edges. If the root is the shared source hub for all transportation requests, then there exists a dynamic programming-based optimal algorithm running in time $O(m^2)$.*

Proof. Without loss of generality, we assume that all the sources are the same at the root. For each run in a star graph, the truck must go from the root to the respective endpoint. If the truck is going to serve another run, it must return to the root. Thus, a truck saves its cost of getting back to the root by $c(s, t_i)$ if this truck delivers its last goods at the destination t_i. An optimal solution OPT will consist of some trucks serving their respective groups, with no runs being shared across groups. For each group, the truck should end at the farthest endpoint within the group, and the order in which the other runs are served does not matter. Sort all the loads as $l_1 \leq l_2 \leq \cdots \leq l_m$. For the first i smallest runs, consider the minimal rental fee R_i truck that must be rented to serve the ith run, along with the prior truck with R_j rental fee to serve the j-th run. Then, truck with fee R_i must serve runs l_{j+1} to l_i. Let $OPT(i)$ denote the minimum rental cost serving the first i transportation requests. We have the following recurrence

$$\begin{aligned} OPT(0) &= 0 \\ OPT(1) &= R_1 + c(s, t_1) \\ OPT(i) &= \min_{j<i} OPT(j) + 2 \sum_{q=j+1}^{i} c(s, t_q) \\ &\quad + R_i - c(s, t_i) \end{aligned} \tag{2.1}$$

$OPT(m)$ is the optimal value. It takes time $O(m \log m)$ to sort the load. For each run $OPT(i)$, $i = 1, 2, \ldots, m$, it takes time $O(m)$ to find the minimum out of all possible j. Since OPT runs m times, the total processing time is $O(m^2)$. □

From the proof of intractability for the shipment rerouting problem, we know that even for an input instance on a line graph, this problem can be NP-hard. However, if we limit that all the transportation requests have the same source-hub or the same destination-hub, then a greedy algorithm is optimal for line graphs and a dynamic programming algorithm is optimal for trees (reduced to star graphs). If we decompose a network into multiple components so that for a component which is *a line graph or a tree with a shared source-hub or a shared destination-hub*, then we have efficient algorithms to get the optimal solution for that component. The idea is similar to the one shown in the proof of Lemma 2.1.

2.2 An integer linear program ALG-IP and a pre-processing decomposition step. In this section, we introduce an integer program that generates the exact solution to the shipment rerouting problem. This program serves as the benchmark compared with our intuitive classical and quantum algorithms. In addition, the formulation is new and we apply some pre-processing decomposition steps to make the input instance smaller, thus, the algorithm based on the integer linear program can run faster.

We remark here that the mixed integer program formulated in [21] only considers the case in which all trucks have identical capacity (and thus, the same rental/operation fee for all the trucks). This assumption in [21] makes the program easy to formulate because one does not need to worry about the combinations of trucks in serving a group of transportation requests.

Consider a complete digraph $F = (H, P)$ with $|H| = n$. Consider m transportation requests $(s_1, t_1, l_1), (s_2, t_2, l_2), \ldots, (s_m, t_m, l_m)$ with K trucks. These K trucks have capacities $L_1 \leq L_2 \leq \cdots \leq L_K$ and their corresponding rental/operation fees are $R_1, R_2, \ldots, R_K$ respectively. We use $c(p(a, b))$ to denote the distance between a hub a to a hub b. We label these trucks as $1, 2, \ldots, K$.

In our solution, we have some truck routes to satisfy all the transportation requests. Such a truck route must start from one source-hub and end at one destination-hub. We only list the hubs that a truck must visit in the graph $F = (H, P)$, and thus, we assume that these truck routes do not share hubs in $F = (H, P)$. Also, we assume these K trucks as K machines (a subset of the trucks are serving the transportation requests). We define the indicator variables in Table 2 and present the constraints over these 0-1 variables in the list from C_1 to C_7. The objective is listed as C_8.

$X_{i,j,p}$	an indicator variable on the source s_i of request (s_i, t_i, l_i) on the jth route at the position p, $X_{i,j,p} = \begin{cases} 1, & \text{if } s_i \text{ is at position } p \text{ on the route } j \\ 0, & \text{otherwise} \end{cases}$
$Y_{i,j,p}$	an indicator variable on the destination t_i of request (s_i, t_i, l_i) on the jth route at the position p, $Y_{i,j,p} = \begin{cases} 1, & \text{if } t_i \text{ is at position } p \text{ on the route } j \\ 0, & \text{otherwise} \end{cases}$
Z_j	an indicator variable on whether the route j is selected or not. $Z_j = \begin{cases} 1, & \text{if the route } j \text{ is selected} \\ 0, & \text{otherwise} \end{cases}$

Table 2: 0-1 variable used for the integer linear program

C_1. A transportation request must be served. The source and the destination of a request must be placed at some positions of the routes.

$$\sum_{p=1}^{2m} \sum_{j=1}^{K} X_{i,j,p} = 1, \qquad i = 1, \ldots, m$$
$$\sum_{p=1}^{2m} \sum_{j=1}^{K} Y_{i,j,p} = 1, \qquad i = 1, \ldots, m$$

C_2. For each position p on a route, we have at most one transportation request.

$$\sum_{i=1}^{m} (X_{i,j,p} + Y_{i,j,p}) \leq 1, \quad j = 1, \ldots, K, p = 1, \ldots, 2m$$

C_3. A request has its source-hub an destination hub on the same route.

$$\sum_{p=1}^{2m} X_{ijp} = \sum_{p=1}^{2m} Y_{ijp}, \quad i = 1, \ldots, m, j = 1, \ldots, K$$

C_4. A request source-hub should be before its destination-hub on the route. This constraint replaces the above one in our formulation.

$$\sum_{p=1}^{q} Y_{ijp} \leq \sum_{p=1}^{q} X_{ijp}, \quad i = 1, \ldots, m, j = 1, \ldots, K$$
$$q = 1, \ldots, 2m$$

C_5. A route is chosen given it having transportation requests on it.

$$Z_j \geq \sum_{p=1}^{2m} X_{ijp} = \sum_{p=1}^{2m} Y_{ijp}, \quad i = 1, \ldots, m, j = 1, \ldots, K$$

C_6. A truck on a route should have its capacity no less than the load it carries along the route.

$$L_j \geq \sum_{p=1}^{q} \left(\sum_{i=1}^{m} (X_{ijp} \cdot l_i) - \sum_{i=1}^{m} (Y_{ijp} \cdot l_i) \right),$$
$$j = 1, \ldots, K, q = 1, \ldots, 2m$$

C_7. The indicator variables X and Y, if they are non-0, should be consecutive in the positions of a route. That is, if a position is not occupied by some source-hub or destination-hub of a transportation request, then its immediate following position should not be occupied by some hub.

$$\sum_{p=1}^{q} \sum_{i=1}^{m} (X_{ijp} + Y_{ijp}) + \sum_{p=1}^{q} \sum_{i=1}^{m} (X_{ijp+1} + Y_{ijp+1})$$
$$\geq 2 \sum_{p=1}^{q} \sum_{i=1}^{m} (X_{ijp+2} + Y_{ijp+2})$$
$$j = 1, \ldots, K, q = 1, \ldots, 2m - 2$$

C_8. The objective is to minimize the total rental fee and the total travel cost.

$$\min \quad \textstyle\sum_{j=1}^{K} R_j \cdot Z_j + \sum_{i=1}^{m} \sum_{i'=1}^{m} c(p(i,i')) \sum_{j=1}^{K} \left(1 + \lfloor \frac{(X_{ij(p-1)} - 1) + (Y_{ijp} - 1)}{2} \rfloor \right)$$

We remark here that the work [21] did not consider the truckload constraints on various orders of source-hubs and destination-hubs as what we did in the above C_4. Our integer linear program is a 0-1 linear program. We also remark that the constraint C_7 ensures that we correctly calculate the minimum cost of traveling along the routes specified in the constraint C_8.

Benders decomposition for the 0-1 linear program. Applying this to real-world networks (as outlined in Section 4), we find out the linear program size is huge but the program has a block diagonal structure which lends itself toward optimization via decomposition. Therefore, we take a pre-processing step to decompose the program into smaller ones with fewer variables. While there exist other well-known decomposition techniques, most notably Dantzig-Wolfe decomposition [5], Benders decomposition [2] is the best-suited decomposition algorithm due to the presence of the linking variable structure column, shown in Figure 1.

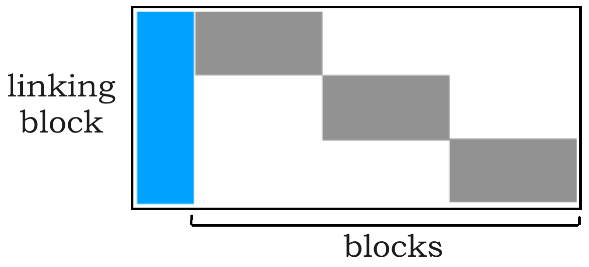


Figure 1: Benders block diagonal coefficient matrix with linking variable structure

Benders decomposition uses a divide-and-conquer framework to solve large-scale convex problems by separating variables into master problems and sub-problems (refer to Figure 1) and it is the core part of a bound-and-price approach [1]. These problems are created by dividing the problem variables between the linking and non-linking variables. Linking variables are stricter and in the case of mixed-integer linear programs are discrete or binary variables used to construct the master problems — this master problem produces the initial solution and lower bound solution to the initial problem. The remaining non-linking variables are looser (such as continuous variables) and are included in the sub-problems. These problems are purely linear programming problems and thus easier to solve. Solving the sub-problems creates additional constraints (known as Benders cuts) used to guide the algorithm towards the optimal solution. Benders decomposition operates by first solving the master problem without any constraints and then solving the sub-problems. Solving a sub-problem can have one of three outcomes:

1. if the sub-problem is unbounded, a Benders cut is added to eliminate unfeasible first-stage solutions,

2. if the sub-problem is unfeasible, then the initial problem is either unfeasible or unbounded,

3. if the sub-problem has a finite optimal solution, a Benders cut is used to improve the objective value.

The master problem is then resolved with these new Benders cuts and the process is repeated until an optimal solution is found or infeasibility is proven. We implement Bender decomposition in Section 4 as one pre-processing step.

2.3 A weighted matching algorithm ALG-WM.

In addition to the 0-1 linear program formulated above, we design a classical algorithm for the shipment rerouting problem. This algorithm is based on our insight into the relationship among the transportation requests: twisted runs and isolated runs. Heuristically, a group of twisted runs have their source-hubs close to each other and their destination-hubs close to each other. For a group of isolated runs, their source-hubs are closer to their respective destination-hubs.

A pre-processing step. In our pre-processing step, we identify the relationship between any two truck runs. However, an optimal solution should specify the order of visiting the source-hubs and/or destination-hubs for multiple truck runs. Avoiding enumerating all the orders, we take a fast and efficient way as specified below to pre-process the graph $G = (H, P)$:

1. Build a graph F' with m vertices initially, with each vertex representing a truck run.
2. For any two truck runs, calculate their relationship (twisted or isolated) as specified in Section 2.1. Two vertices in F' have an edge if and only if they are twisted.
3. Find the connected component of the graph. Each connected component represents a *super-twisted-run.* Two super-twisted runs are labeled as *isolated.*

Calculating the twisted/isolated relationship takes time $O(m^2)$ and the breath-search-first algorithm calculating the connected components takes time $O(m)$. We thus have the following result.

Lemma 2.2. *The pre-processing step takes time $O(m^2)$.*

A layered algorithmic approach with two stages. We now extend the concept of twisted runs and isolated runs to super-twisted runs and super-isolated runs respectively, calculated in the pre-processing step. We apply the heuristics on twisted runs and isolated runs on the super twisted run and the super isolated runs. In this way, we have a layered approach in assigning routes to the transportation needs as sketched below:

1. In this first phase: For each super twisted run, design a routing algorithm to route all the twisted runs in it.

   Repeat doing so until all the runs within a super-twisted run are scheduled.
2. In the second phase: For all the isolated runs, design a routes algorithm to route all the super-isolated runs.

A weighted matching algorithm for each stage. Consider a group of twisted runs in a super-twisted run. If the super-twisted run's size is small (as a constant), then we find an optimal sequence of scheduling these runs can be done by using a brute force approach. If a super-twisted run's size is large, then we use a greedy algorithm to schedule all the runs. This greedy algorithm depends on a concept *marginal cost.*

Definition 1. (Marginal (merging) cost.) *Consider two (super)runs. The* marginal cost *is the minimum difference between an optimal cost run for these two and the sum of the separate two runs.*

Given two transportation requests (s, t, l) and (s', t', l'), the marginal cost is the minimum value in the 3rd column in Table 1.

In each phase, use a weighted matching algorithm to find out the order of scheduling the (super)runs to reduce the total cost incurred given the marginal cost is negative. The algorithm works as below:

1. In this first phase: Consider each of the super-twisted runs calculated in the pre-processing step. For each two runs, use an edge to connect them and use the marginal cost, if negative, to represent the edge's value.

   Apply the weighted matching algorithm [7] to merge the runs and recalculate the marginal costs for those merged runs. Repeat the above step until the merging step stops (with no negative-marginal-cost edges connecting runs).
2. In the second phase: Regard each of the super-isolated runs calculated (with each super-twisted relabeled as a super-isolated run after the first phase).

   Apply the weighted matching algorithm [7] to merge the runs and recalculate the marginal costs for those merged runs. Repeat the above step until the merging step stops (with no negative-marginal-cost edges connecting runs).

Initially, we have m runs and $O(m^2)$ edges connecting some of them. The weighted matching algorithm [7] takes time $O(m^2)$ for each round of merging and the number of runs is reduced to half in each round. We have at most $O(\log m)$ rounds of merging. Therefore, the total running time of the matching algorithm (along with the cost of calculating the marginal cost) is bounded by

$$O\left(m^2 + \left(\frac{m}{2}\right)^2 + \left(\frac{m}{2^2}\right)^2 + \cdots\right) = O\left(m^2\right)$$

3 An Quantum Algorithm based on QUBO Representation

In this section, we design a quantum algorithm for the shipment rerouting problem. Though the 0-1 integer program formulated in Section 2 can be solved by most classical commercial solvers to get an optimal solution, such a program is not compatible with any quantum algorithmic solvers known in the current noisy intermediate-scale quantum (NISQ) era. All known quantum annealing-based solvers only accept problems that have quadratic unconstrained binary optimization (QUBO) solutions. These problems are of the form $\min \mathbf{x}^T\mathbf{Q}\mathbf{x}$, where $\mathbf{x} \in \{0,1\}^n$ and $\mathbf{Q}$ is an upper triangular matrix with each entry Q_{ij} representing the pairwise weight of $x_i x_j$.

Note that there exist various practical approaches of integer-to-binary mapping for quantum annealers (see [11] and the reference therein). The 0-1 linear program that we have developed in Section 2.2 can be converted into a QUBO efficiently, through a 1-to-1 mapping between QUBO and Ising models using spin-binary bijection. The challenges here are (1) how to use fewer qubits for the QUBO, and (2) how to improve a solution's precision.

On reducing the number of qubits used in the QUBO formulation. We pre-process the input instance through the Bender decomposition so that the input instance can be decomposed into multiple parts with each part having a smaller size. In addition, we use the pre-processing step to calculate the twisted runs and isolated runs so that we can identify more variables' values in the constraints before we encode these variables into QUBO. Both pre-processing steps have been described in Section 2.1 and Section 2.2. The kernel of the 0-1 linear program, which is rewritten as a QUBO formulation with multiple decomposed parts, can be solved via quantum search methods, or heuristically through quantum annealing approaches [17].

On alternatives of solving the 0-1 integer program. We use the quantum version of linear program solver [12] to get an approximation and then use exact quantum branch-and-bound quantum algorithms [15, 3] to find the optimal solution. These branch-and-bound quantum algorithms are essentially quantum backtracking variants, better than naively using Grover's search algorithm [9, 10]. In addition, the solution's precision depends on Grover's algorithm [9, 10] (which is used to find the minimum value) implemented for the error-tolerant quantum hardware in the current NISQ era. Note that in our experiments, we do not include these alternative quantum algorithms as the errors and the overheads introduced prohibit us from getting some reasonable solutions to the optimal one.

4 Experiments

In this section, we design experiments to compare the classical algorithms and the quantum algorithms on real-life data sets.

4.1 Settings. The code is available at `https://shorturl.at/ED5Ww`.

On input data. The input data instance comes from the Transportation Networks Github [18]. This data repository was initially developed to solve the traffic assignment problem [16], which the shipment rerouting problem greatly resembles. The algorithms mentioned in Section 2 and Section 3 are tested across 10 shipment input sizes, with each over the following 5 networks: *Chicago*, *Barcelona*, *Winnipeg*, *Anaheim*, and *Eastern Massachusetts*, described in Table 3.

names	# of nodes	# of edges
Eastern Massachusetts	74	258
Anaheim	416	914
Chicago	933	2950
Winnipeg	1052	2836
Barcelona	1020	2552

Table 3: Testbed: 5 networks

Given an underlying network, the truck runs are randomly generated, and their candidate routes are created by conducting depth-first traversals from each run's source to its destination. As there could be exponentially many candidate routes for each run, we limit each shipment to five candidate routes.

We conduct two sets of tests: one considering small input sizes of 30, 50, 80, and 100 runs and the other considering exponentially large input sizes of 32, 64, 128, 256, 512, and 1024. The large-size input instances require the installation of IBM CPLEX Optimization Studio.

On hardware. We record the running time and the solution quality for each test. The running time is measured in minutes and seconds. The classical algorithms are implemented using Python and they run on a CPU using a 12th Gen Intel Core i7-1260P processor with 16.0 GB memory.

The quantum algorithm model as a binary quadratic model uses PyQUBO [22, 19] and it runs using D-Wave's Leap hybrid solver. The Leap hybrid solvers are proprietary quantum-classical-quantum hybrid solvers offered by D-Wave to solve large QUBO problems on quantum processing units (QPUs). As the problems are submitted to remote solvers, wall clock measurement is not sufficient. In the experiments, the runtime of quantum algorithms is queried using Ocean API.

4.2 Experimental results and analysis. Each algorithm has been tested 10 times for each input on all the networks with the results displayed in the plots. All algorithms have a similar degree of optimality and thus, we only report these algorithms' runtime to differentiate them.

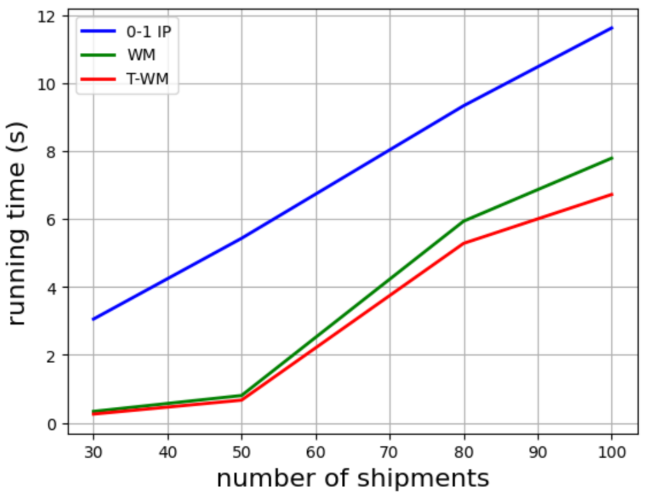


Figure 2: Running time for small instances for Chicago

From these plots, we have the following observations and the summary of our conclusions is in Figure 14.

1. Across all five networks, ALG-IP had the worst performance across *small-size inputs* while performing significantly better on the *larger-size* inputs.

   Part of the reason is that most of the variables are generated from the initial graph. The additional shipments made little difference in the running time for this problem as opposed to the other algorithms. For this reason across all tests, ALG-IP displays a clear linear growth rate.

2. On the other hand, ALG-WM and ALG-WM with twisted groups perform well across all small inputs but do poorly on larger-size inputs.

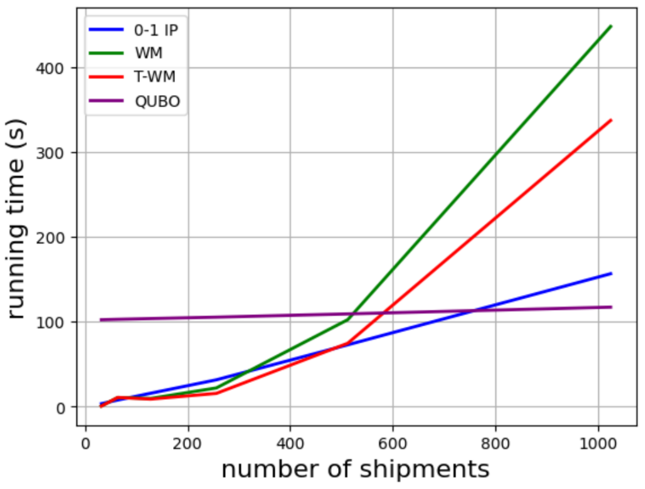


Figure 3: Running time for large instances for Chicago

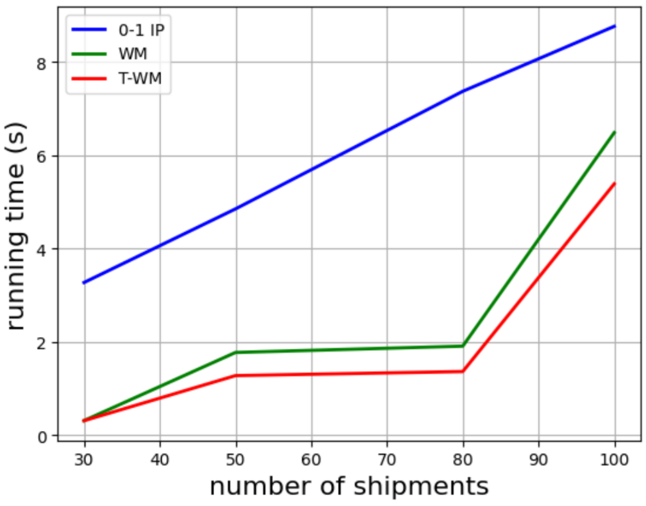


Figure 4: Running time for small instances for Anaheim

   On nearly all tests, the algorithm ALG-WM with twisted groups performed just as well or better than ALG-WM. The difference in performance grows with the number of shipments scheduled. On most networks, this gap grows in size after 80 runs. This observation agrees with our insight on twisted runs and isolated runs.

3. While the quantum algorithm has a high overhead cost depending on the size of the graph, it does not grow very quickly appearing to be near constant.

   Part of the reason is that in the QUBO formulation, most of the variables come from the initial graph, thus adding more runs can only make little difference in the scales of the QUBO problem.

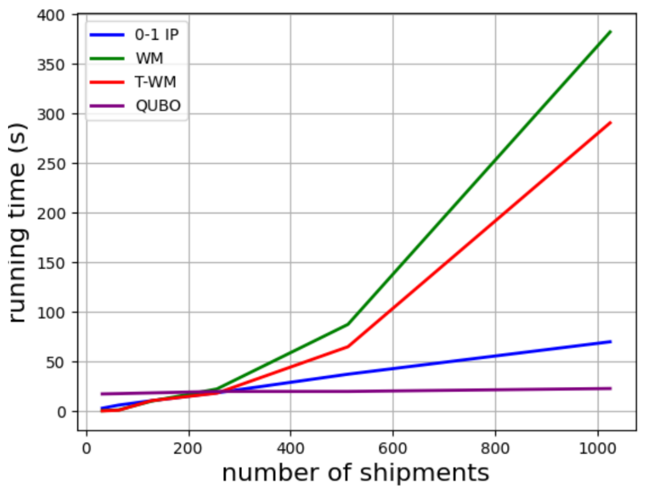


Figure 5: Running time for large instances for Anaheim

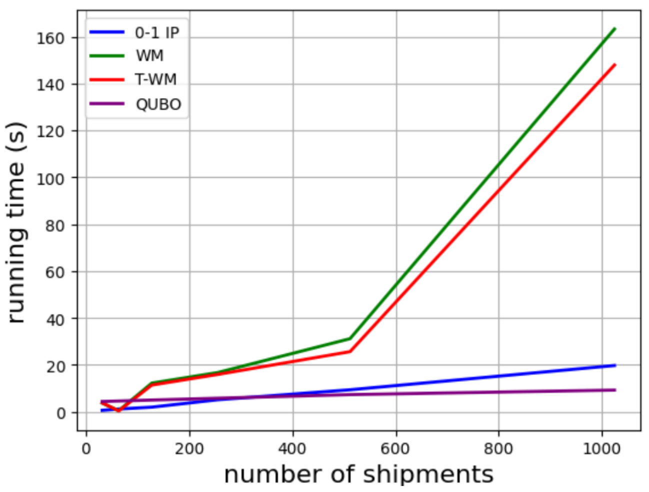


Figure 7: Running time for large instances for Eastern Massachusetts

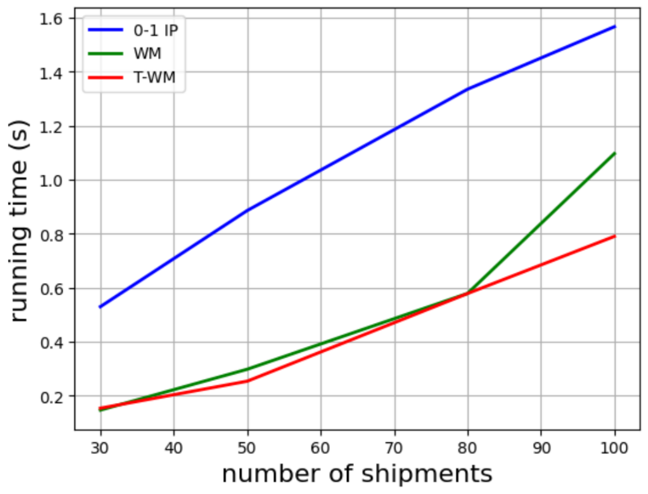


Figure 6: Running time for small instances for Eastern Massachusetts

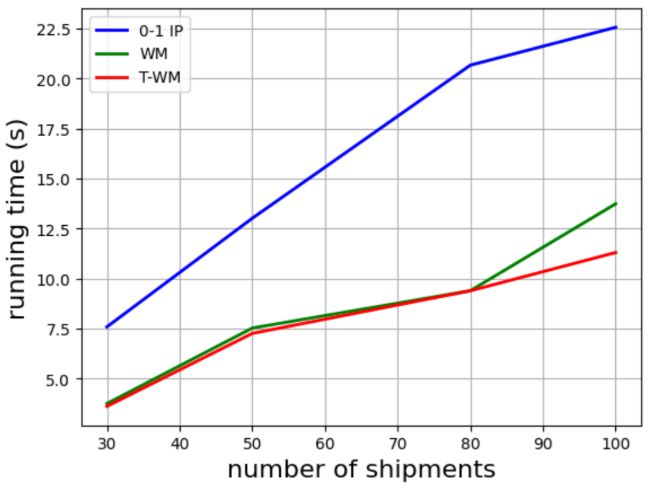


Figure 8: Running time for small instances for Winnipeg

4. For all the algorithms over various networks, we observe that increasing the number of hubs of the input graph can shift to the point that the quantum algorithm and ALG-IP outperform the ALG-WM with twisted runs, in terms of running time.

On algorithm selection. To capture an algorithm's suitability on various input instances, we define a test's *shipment density*, the ratio between the number of shipments and the number of nodes in the overlaying network. We observe that the tests with high shipment densities generate more bender cuts (Figure 13) and thus for such input instances, algorithms converge to the optimal solutions quickly. That is to say, for graphs with 'denser' runs, ALG-IP and the quantum algorithm run faster. At the same time, the opposite effect occurs over the tests with low shipment densities. They generate fewer Benders cuts and converge more slowly. Note that we discover that the quantum algorithm performs far slower than the classical algorithms over the small input sizes, and thus was omitted from the graph for better presentation.

5 Conclusions

In this paper, we consider the shipment rerouting problem. We give two classical algorithms, a 0-1 integer linear program and a weighted matching algorithm based on a key concept among transportation requests. The concepts of twisted runs and isolated runs lead to fast scalable computing for small/medium or sparse input instances. Our next step of research is to develop

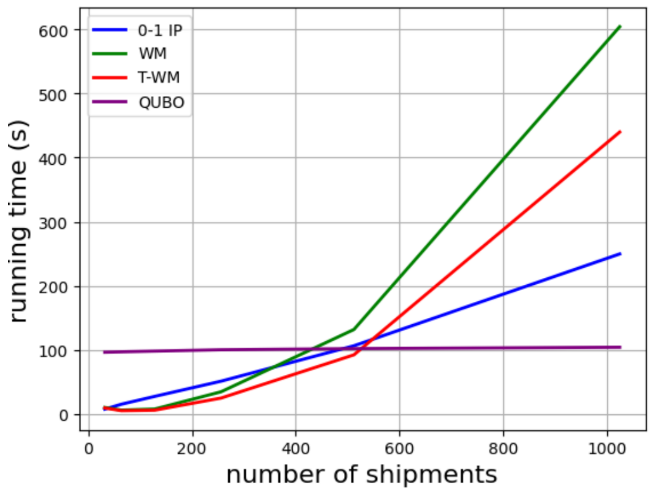


Figure 9: Running time for large instances for Winnipeg

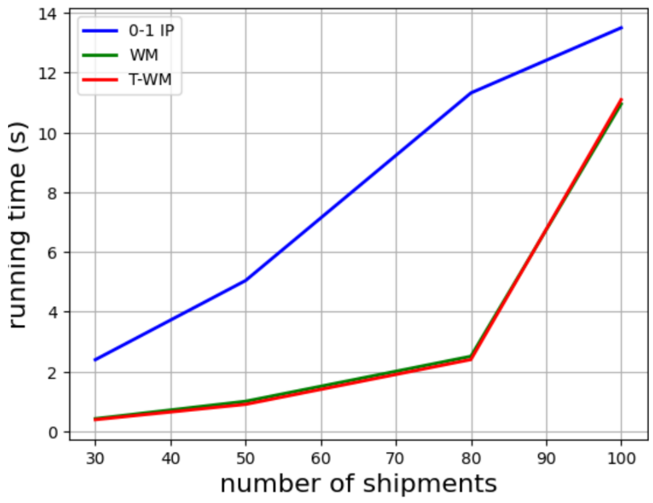


Figure 10: Running time for small instances for Barcelona

fast quantum algorithms for this problem. Motivated by the fact that the branch-and-bound quantum algorithms [15, 3] have been studied and in some cases yield running times that are substantially better than naively using the Grover's algorithm, we design a variational quantum algorithm for the branch-and-price framework. We are going to compare this quantum approximation optimization algorithm against the proposed solutions in this work.

References

[1] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-price: Column generation for

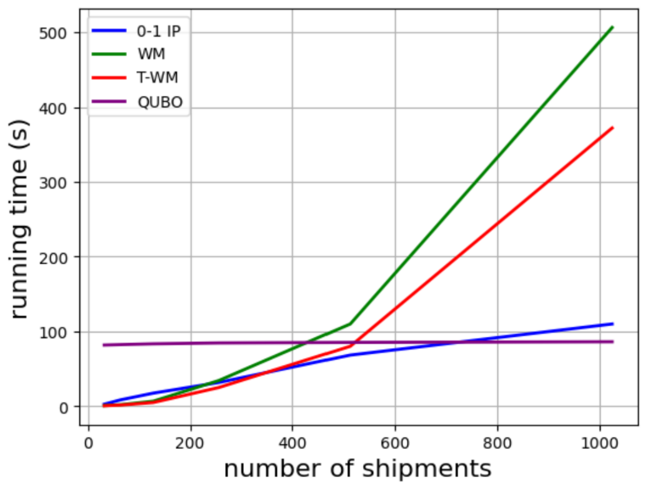


Figure 11: Running time for large instances for Barcelona

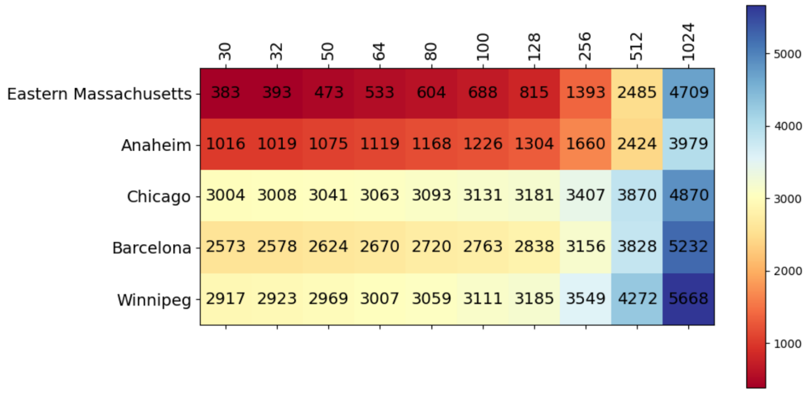


Figure 12: Heatmap of IP and QUBO's variable counts over various tests

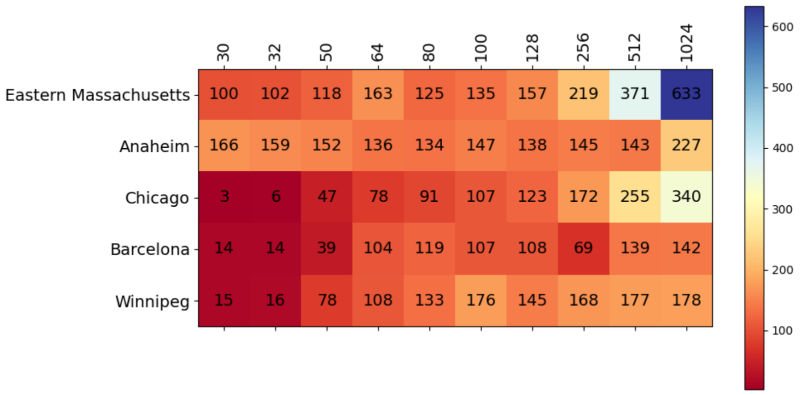
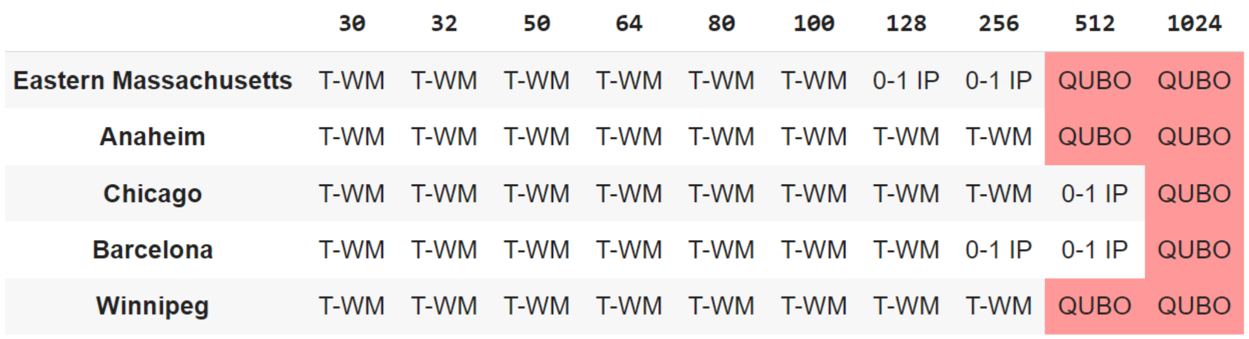


Figure 13: Heatmap of the number of Benders cuts over various tests

solving huge integer programs. *Operations research*,



	30	32	50	64	80	100	128	256	512	1024
Eastern Massachusetts	T-WM	T-WM	T-WM	T-WM	T-WM	T-WM	0-1 IP	0-1 IP	QUBO	QUBO
Anaheim	T-WM	T-WM	T-WM	T-WM	T-WM	T-WM	T-WM	T-WM	QUBO	QUBO
Chicago	T-WM	T-WM	T-WM	T-WM	T-WM	T-WM	T-WM	T-WM	0-1 IP	QUBO
Barcelona	T-WM	T-WM	T-WM	T-WM	T-WM	T-WM	T-WM	0-1 IP	0-1 IP	QUBO
Winnipeg	T-WM	T-WM	T-WM	T-WM	T-WM	T-WM	T-WM	T-WM	QUBO	QUBO

Figure 14: Algorithm selection for various shipment input sizes. T-WM represents the twisted ALG-WM algorithm; 0-1 IP represents the algorithm ALG-IP; QUBO represents our quantum algorithm

46(3):316–329, 1998.

[2] Jacques F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[3] S. Chakrabarti, P. Minssen, R. Yalovetzky, and M. Pistoia. Universal quantum speedup for branch- and-bound, branch-and-cut, and tree-search algorithms. *arXiv:2210.03210*, 2022.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 4th edition, 2022.

[5] George B. Dantzig and P. Wolfe. Decomposition principle for linear programming. *Econometrica*, 9:767–778, 1961.

[6] Yongcheng Ding, Xi Chen, Lucas Lamata, and Enrique Solano Mikel Sanz. Implementation of a hybrid classical-quantum annealing algorithm for logistic network design. *SN Computer Science*, 2(2), 2021.

[7] Harold N. Gabow and Robert Endre Tarjan. Faster scaling algorithms for general graph- matching problems. *Journal of the ACM (JACM)*, 38(4):815–853, 1991.

[8] Michael R. Garey and David S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-completeness*. W. H. Freeman, 1979.

[9] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*, pages 212–219, 1996.

[10] Lov K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Physics Review Letters*, 79(2):325–328, 1997.

[11] Sahar Karimi and Pooya Ronagh. Practical integer-to-binary mapping for quantum annealers. *Quantum Information Processing*, 18(Article 94):1–24, 2019.

[12] Iordanis Kerenidis and Anupam Prakash. A quantum interior point method for LPs and SDPs. *ACM Transactions on Quantum Computing*, 1(Article No. 5):1–32, 2020.

[13] E. L. Lawler, Jan Karel Lenstra, A. H. G. Rinnooy Kan, and David B. Shmoys. *The traveling salesman problem: a guide tour of combinatorial optimization*. Wiley, 1985.

[14] Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 11(2):208–230, 2014.

[15] Ashley Montanaro. Quantum speedup of branch-and-bound algorithms. *Physics Review Research*, 2(013056):1–23, 2020.

[16] Michael Patriksson. The traffic assignment problem : models & methods. 2015.

[17] Escolástico Sánchez Samuel Fernández-Lorenzo Jorge Luis-Hita Enrique Lizaso Samuel Mugel, Carlos Kuchkovsky and Román Orús. Dynamic portfolio optimization with real datasets using quantum processors and quantum-inspired tensor networks. *Physics Review Research*, 4(1):013006, 2022.

[18] Ben Stabler, Hillal Bar-Gera, and Elizabeth Sal. Transportation networks for research. `https://github.com/bstabler/TransportationNetworks/`. accessed on July 15, 2024.

[19] Kotaro Tanahashi, Shinichi Takayanagi, Tomomitsu Motohashi, and Shu Tanaka. Application of ising machines and a software development for ising machines. *Journal of the Physical Society of Japan*, 88(6):061010, 2019.

[20] Fei-Fei Yan, Zhen-Peng Xu, Qiang Li, Jun-Feng Wang, Ji-Yang Zhou, Wu-Xi Lin, Jin-Shi Xu, Yuyi Wang, Chuan-Feng Li, and Guang-Can Guo. Room-temperature implementation of the quantum streaming algorithm in a single solid-state spin qubit. *Physical Review Applied*, 16(024027), 2021.

[21] Sheir Yarkoni, Andreas Huck, Hanno Schulldorf, Benjamin Speitkamp, Marc Shakory Tabrizi, Martin Leib, Thomas Back, and Florian Neukart. Solving the shipment rerouting problem with quantum optimization techniques. In *Proceedings of the International Conference on Computational Logistics (ICCL)*, pages 502–517, 2021.

[22] Mashiyat Zaman, Kotaro Tanahashi, and Shu Tanaka. PyQUBO: Python library for mapping combinatorial optimization problems to QUBO form. `https://arxiv.org/abs/2103.01708`.