CHAPTER 5

Hybrid quantum algorithms for real-world optimization

While the potential benefits of quantum computing and quantum annealing are well-motivated, it is clear from the previous sections that there are several limitations when trying to solve optimization problems directly in current-generation QPUs. In recent years, classical software development has complemented the quantum hardware in an attempt to bridge the gap between the restrictive quantum hardware and more real-world applications of quantum annealing. One of the proposed ways to use quantum annealing in practice is the concept of *hybrid quantum-classical* algorithms. With these, algorithms are constructed so that the quantum annealing QPU is used in some way by an inner loop (as a sub-solver/sampler, a mutation step in a genetic algorithm, etc.), thus offloading some difficult critical task to the QPU. These kinds of algorithms have been used in the past to tackle arbitrarily-structured optimization problems, in an attempt to solve larger and more realistic combinatorial optimization problems [37, 38, 39]. To support the development of these algorithms, D-Wave Systems released a Python package dedicated to constructing such hybrid algorithms, called `dwave-hybrid` [99]. In this chapter we investigate the construction and use of hybrid algorithms using these tools, as well as the black-box hybrid optimization algorithm provided by D-Wave System. We motivate and contextualize their use through real-world combinatorial optimization problems, and build custom optimization routines to solve QUBO/Ising problems in real-time.

5.1 Motivating a real-world traffic optimization use-case

One of the first real-world applications demonstrated for quantum annealing in practice was traffic flow optimization [98]. This was motivated by the increased focus on autonomous driving, smart cities/infrastructure, and the need for cutting-edge computational resources to handle the complex task. In order to test this in practice, a pilot project was conducted in which the QUBO outlined in [98] was solved in real-time to navigate a fleet of buses, providing a turn-by-turn navigation service for the Web Summit 2019 conference in Lisbon, Portugal. The motivation for this project comes from observations in the automotive industry, that as cities around the world continue to grow in both size and population, traffic congestion becomes an increasingly prevalent problem [121]. This is especially apparent during events that congregate large numbers of people for specific periods of time. For example, conferences, sporting events, and festivals can cause temporary but significant disruption to the cities' transportation systems, resulting in delays for the residents of those cities [122, 123]. A key issue is that permanent transportation infrastructure, such as rail lines or roads, are costly and slow to modify given the temporary nature of these events. In light of this, the advent of smart traffic management systems offers possible improvements to existing transportation systems with minimal overhead in regards to implementation. Some requirements for such systems include the management of the mobility flows in real- or near to real-time using flexible and modular software components. The goal of this project was therefore meant to address two very specific questions related to applying quantum annealing in a real-world scenario:

1. How do we design customized bus routes to avoid traffic congestion during big events?
2. How would one build a real-time production application using a quantum processor to manage such a traffic system?

In order to address both of these questions, we separate the work into two separate phases: the first is concerned with understanding the input required for such a live navigation service to run in practice, and the second phase presents the construction of hybrid optimization service that complies with the demands of the first. Concretely, the goal is to adapt the model presented in [98] to a live setting, and test the assumption that using quantum annealing to avoid overlaps between vehicle routes reduces traffic congestion. Thus, we are able to construct meaningful optimization problems in an application setting, and address the application needs using hybrid quantum techniques.

To answer the first question, we needed to consider real-world road networks and movement data as input to our navigation service. This required us to find and fix (possibly multiple) start and endpoints of a custom bus service, where each bus had its individual route customized by considering all other buses in the fleet in real-time, as described in detail in [98]. By analyzing origin/destination (OD) matrices of peoples' movement data, detailing the volume of movement streams from the Web Summit conference venue (Altice arena) to different zones of the city of Lisbon on an hourly basis, we were able to identify locations of interest for our navigation service. The results showed a total of 225 OD matrices in a study area of 93 zones throughout Lisbon, shown in Figure 5.1. This data included demographic census data, mobility behavior from surveys, Lisbon traffic counts, floating car data, mode of choice and network models from the city, for both dates within and outside the Web Summit time frame.

Three express bus lines were proposed to serve the demand from the selected zones to the Web Summit: a red, green, and blue line, with a total of 23 bus stops. One express line was dedicated for the return traffic from the conference venue to the city center (the black line). The red and green bus lines operated only during the morning period, whereas the black and blue lines operated both in the morning and evening. Bus departures were scheduled every 30 minutes for all lines. For the morning, two lines (red and green) covered the demand in the northern part of the city center, picking up visitors along 7 dedicated bus stops and meeting at the farthest point of the line at the Saldahna roundabout. From this point to Web Summit, the visitors were no longer picked up and the bus was navigated solely using the quantum navigation service. For the black and blue lines, the portion of the routes between the Web Summit and Alameda station were navigated using the quantum navigation service.

To determine the schedules of the fleet, we identified the peak traffic demand in the selected zones was from 09:00-10:00 and 10:00-11:00, with 6314 trips towards Web Summit. For the evening demand (the return trips to the city center), peak hours were from 16:00-17:00 and 17:00-18:00 with 7600 trips leaving the Web Summit.

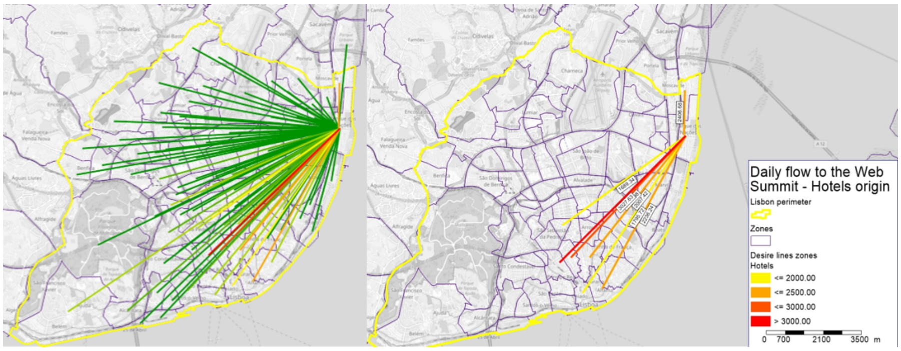

Figure 5.1: Left: Origin-destination matrices from the Web Summit conference to the city center. Right: Selected OD matrices for hotel- and room rental-related trips. Visualization provided by PTV Visum software.

The results also highlighted that the zone with the lowest public transport usage was zone 75 (Santa Maria Maior-Castelo), with 45% of total trips being public transit. This indicated that the modal split in this zone can be heavily improved relative to other zones with higher average public transit usage (65% and above). Given the performed analysis for each of the four selected zones, and considering the estimated demand in both the morning and evening, a total of 9 buses was proposed to Carris as a recommendation for the Quantum Shuttle fleet volume.

5.2 Building a quantum optimization service

Building a functioning web service that uses a quantum processor while providing meaningful navigation optimization imposes a specific set of both conditions and constraints. By setting the goal of navigating buses in real-time, we require that a live connection between three different services be consolidated simultaneously– the navigation app (run on an Android tablet), the traffic data, and the hybrid quantum optimization. We briefly describe the content and scope of each component, then explain how they were combined in the final optimization service, which we call the Quantum Web Service (QWS).

5.2.1 Bus navigation Android app

Development of the Android app was divided into two parts: the front-end (visualization for the bus drivers) and the back-end (server-side communication to the QWS). We explain separately how the two parts were implemented, then consolidated, to satisfy the demands of the navigation service. The front-end development was outsourced to an external supplier, while the back-end optimization (including the API to access it and database management) was developed and implemented internally.

Front-end. The front-end of the bus navigation comprised of an Android application, visualizing the turn-by-turn navigation, operated on an Android tablet. The main role of the app was to plot the custom routes provided by the QWS and initiate turn-by-turn navigation with voice instructions. Additional functionality was built into the app to allow bus drivers to start/stop the current and next trips they were meant to follow, as well as to track the current location of the buses in the fleet to allow for the optimization of the route selection. Trips were also ended automatically once a vehicle was within 15 meters of its defined destination. The visualization portion of the app was built using the Mapbox SDK[1] to render the custom routes obtained by the QWS.

Back-end. The back-end of the navigation app acted as an interface between the Android app and the QWS. The back-end was used to send the collected data of all the vehicles in the fleet at fixed time intervals to the QWS over HTTP. Responses from the QWS were also distributed by the back-end to their respective vehicles.

The most important role of the back-end was to keep the data flow in sync between the vehicles and the QWS, as both synchronous and asynchronous communication protocols were used. This was necessary in order to properly construct correct traffic-flow QUBOs which represented the live navigation conditions. The QWS was designed in such a way that it accepted a single consolidated request consisting of the location information for all the vehicles, both live and projected. It was the role of the back-end software to consolidate all the locations it received from all the vehicle devices at different frequencies, and send it to the QWS. This meant that the back-end maintained an index of each vehicle's request and response. Because of

[1] https://www.mapbox.com/

the different requirements of the live location tracking and the projected locations, the requests submitted by the back-end to the QWS were separated between two destinations on the QWS: /update and /optimize. The /update request submitted the live location of each Quantum Shuttle vehicle to the QWS on a 30 second interval, while the projected location (and in return the customized route returned) was submitted to the /optimize URL at an interval of 120 seconds. The necessity of splitting the requests to separate URLs and their respective timing was discovered during the testing phase, explained in Section 5.3. Lastly, the back-end was also responsible for determining the difference between two subsequent optimized routes for the same vehicle.

5.2.2 Constructing live traffic-flow QUBOs from data

The approach used for custom navigation of the vehicles followed an approach based on [98]. At the start of every trip and at regular time intervals until completion of each trip, multiple candidate routes needed to be generated between the current location of each bus in the system to its assigned destination. These routes also needed to be traffic-aware to reflect the current conditions of the city road network. To accomplish this, we used a live traffic services provider, HERE Technologies [1]. Using their routing API [2], we were able to generate between 3-5 traffic-aware candidate routes per vehicle at every optimization step with minimal overhead. It is important to note that since the vehicles were operated in parallel in different parts of Lisbon, different routes were likely to be suggested for each vehicle in the system at every optimization step. Often, however, subsets of these suggested routes overlapped, necessitating the optimization of the routes' selection to minimize congestion. In this scenario, identical GPS points were returned from the HERE API describing the shape of the overlapping portion of the routes. Therefore, the GPS points were used directly to form the optimization problem, instead of the road segments as in [98]. The QUBO formulation of the objective function is then:

$$\mathrm{Obj} = \sum_{p \in P} \mathrm{cost}(p) + \lambda \sum_{i} \left(\sum_{j} q_{ij} - 1 \right)^{2}, \tag{5.1}$$

[1] https://www.here.com/

[2] https://developer.here.com/

where q_{ij} are the binary decision variables associated with vehicle i taking route j, P is the set of all GPS points that overlap in the suggested routes, λ is a scaling factor ensuring only one route is selected per vehicle in the QUBO minimum, and cost(p) is the cost function associated with each GPS point p in P:

$$\mathrm{cost}(p) = \left(\sum_{q_{ij} \in B(p)} q_{ij} \right)^2 . \tag{5.2}$$

Here, q_{ij} is as before, and $B(p)$ is the set of all binary variables that contain the GPS point p. Thus, the final selection of routes in the optimum of the QUBO represent the routes that minimally overlap with all other selected routes.

5.2.3 Solving the traffic-flow QUBOs with quantum annealing

For live traffic navigation, our quantum optimization service needed to meet specific conditions. Because of the time-sensitive nature of traffic navigation, our solution needed to respond with valid solutions to the optimization problem quickly. The algorithm also needed to handle varying sizes and complexity of the traffic-flow optimization problem, as it needed to optimize the route selections of the vehicles automatically at regular intervals. We used the D-Wave 2000Q QPU and its respective software stack to deploy our solution on quantum hardware. Three different methods of using quantum annealing was tested to solve the traffic flow optimization QUBOs. We briefly explain each method, how it was implemented, and evaluate them based on our navigation application.

Direct embedding. The most straightforward approach to solving the QUBOs is by minor embedding the graph directly to the topology of the QPU. The benefit of this approach is speed– even with the overhead of transforming the traffic-flow problem to a QPU-compatible graph, using the QPU at the fastest annealing time ($1\mu s$ to obtain a single sample) still returned valid solutions to the problem. The minor-embedding process can be performed on the order of tens or hundreds of milliseconds for small-sized problems. However, the drawbacks of this approach have already been explored and presented in detail in Ch. 2. During the development phase of the Web Summit project we tested various configurations of the direct embedding approach, and found that this method was suitable for up to 10 cars with 5 routes each.

Custom hybrid algorithms. One of the primary goals of this project was to assess whether hybrid quantum algorithms could assist in solving QUBOs in practice. The advantage of this approach is that the use of the QPU in the algorithm can be tailored to the QPU's strength, making better use of a limited resource. However, run-time is sacrificed in waiting for the QPU's response to continue the iterative classical procedure. The original traffic-flow work in [98] employed a hybrid algorithm that used a 64 variable fully-connected graph as the inner loop for the QPU to optimize sub-problems [37]. However, as mentioned previously, minor-embedding dense problems significantly degrades the QPUs performance, which in this case still came at the cost of waiting for the QPU responses. Furthermore, new embeddings needed to be generated for each problem being submitted. In light of this, we developed a custom hybrid algorithm to make better use of the QPU in a timely manner. Our algorithm performed the same Tabu search in the outer classical loop as in [37], but instead of using a single 64 variable sub-problem, we found natural sub-graphs within the TFO problem that were already Chimera structured, thus circumventing the embedding issue. We were able to deploy a hybrid algorithm similar to [39] but without employing chains in the sub-problem. Our method allowed us to increase the throughput of sub-problems to the QPU. However, due to time constraints, we were not able to parallelize the implementation so that it could continually run independent of the request/response portion of the QWS. Therefore, the hybrid algorithm was restarted every time a new route optimization was requested. This incurred significant overhead time, delaying the response to an unacceptable level for live use.

Hybrid Solver Service. As part of D-Wave's online cloud service, in addition to direct QPU access, a state-of-the-art hybrid algorithm is also offered. This service, named the Hybrid Solver Service (HSS), is tailored to solve large, arbitrarily structured QUBOs with up to 10,000 variables. The disadvantage of this method is that we cannot control the exact method with which the QPU is used, instead we use the HSS as an optimization black-box. Access to the HSS was provided through the same API as the D-Wave QPU, which allowed us to integrate it in to the QWS in a modular way. By offloading the overhead associated with starting the hybrid algorithm to the D-Wave remote server, we were able to reduce the response time significantly compared to the other approaches. That the HSS can solve problems significantly larger than the QPU also allows us to seamlessly scale up our QWS to handle hundreds of vehicles with tens of route options for future applications. The HSS provided the best compromise of speed, accessibility, and performance, and was used for the live navigation service during the Web Summit conference.

5.2.4 Creating the live navigation service

We now present an overview of the navigation service as a whole, detailing the various components comprising of the system and how they communicate, for review and clarity. A diagram of the system is presented in Figure 5.2. The components are color-coded according to their roles in the service: blue components relate to data-processing steps, which can be considered as input to the hybrid optimization routine. Gray components comprise the core of the QUBO construction and optimization to solve the traffic-flow problem. The white components are black-boxes provided by external parties.

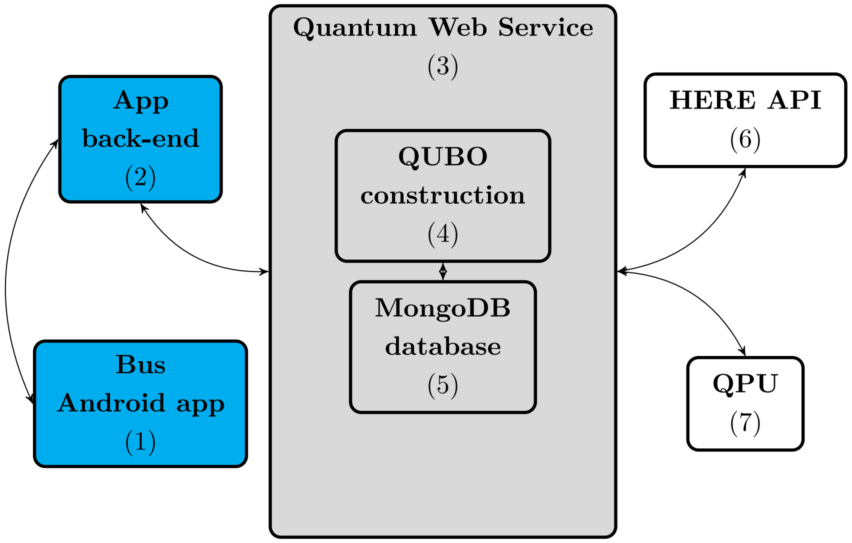

Figure 5.2: Diagram detailing the QWS and its interactions with the other components in the hybrid navigation system, as per the text.

Blue components.

- Component (1), the front-end Android tablet application, provided visual turn-by-turn navigation with vocal instructions to the Quantum Shuttle bus fleet and their drivers. The bus locations were sent every 30 seconds to Component (2).

- Component (2) was the Android app back-end, and submitted the POST requests to the QWS, Component (3). Component (2) consolidated the locations of the vehicles and submitted them to (3) via the /update URL every 30 seconds; this component also formed the POST request to the Component (3) /optimize URL for the route optimization, interpreted the response, and sent the new routes back to (1).

Gray components.

- Component (3) is the framework hosted on AWS (exposed using Flask[1]) and served as the central consolidation point of the other components. In the event of an /update POST request from (2), the QWS updated Component (5), the MongoDB database used to store the data. In the event of an /optimize POST request from Component (2), Component (6) was accessed to request the suggest routes, and that data was passed to Component (4) to construct the traffic flow optimization QUBO problem, then stored in the database using (5). The QUBO was submitted to the D-Wave HSS, Component (7). Component (3) then interpreted the results of the optimization, stored them via (5), and pushed the selected routes back to (2). The API used to communicate with the QWS was custom-built for this application.
- Component (4) is the Python module that implemented the traffic flow optimization QUBO formulation described in Section 5.2.2.
- Component (5) is the Python module wrapped around the MongoDB database; accessing and writing data from the QWS to the database was performed by this component.

White components.

- Component (6) is the HERE Technologies traffic/routing API.
- Component (7) is the D-Wave HSS (or other QPU-based services), accessible via HTTPS.

[1]Flask is a minimalist Python framework for making web-apps, and can be found at: https://www.palletsprojects.com/p/flask/.

5.3 Deploying a quantum optimization service

In preparation for the Web Summit conference, multiple trial runs were performed in both Wolfsburg, Germany (Volkswagen AG headquarters) and Lisbon, Portugal. In this section we describe the two different test scenarios and the lessons learned from each.

5.3.1 Initial tests: Wolfsburg

The first testing phase occurred in Wolfsburg in August 2019, where a small number of cars (1-3) were driven between various origin/destination pairs using the Android app. The goal of this testing phase was to ensure the proper integration of the Android navigation app, QWS API, and the MongoDB database used to keep track of active trips and record results. During testing, several key observations were made, which were used to modify the system.

Location tracking. As mentioned in Section 5.2, it was necessary to both track the live positions of the vehicles, as well as specify a new “origin” per vehicle for every optimization request. In order to ensure that the locations used for optimization didn’t result in infeasible or unrealistic route selections, a projected location was used for each vehicle. Specifically, given a route in the form of a sequence of GPS coordinates, a GPS point further along along the route was passed as the origin for the next optimization step. During the initial testing phase, this position was also used to track the locations of the vehicles. However, this proved to be problematic, since the update frequency was faster than the change in the projected location, making it impossible to track the locations of the vehicles. To solve this, the live location and the location used to determine the new routes were separated: location updates were sent to a different URL independently from the optimization requests every 30 seconds, with the live locations now stored server-side after every update.

Optimization interval. Before testing, the route optimization occurred every 60 seconds. During testing it was observed that this frequency was too fast, resulting in different routes being assigned to the vehicles every time an optimization problem was solved. Furthermore it was observed that, given a new route A after the optimization, a *different new route* B would sometimes be suggested while navigating to route A. This would have made navigating buses with real passengers impractical. It was discovered that this phenomenon occurred due to two main reasons: firstly, quantum annealing and other such hybrid algorithms are metaheuristic optimization algorithms, meaning they are not guaranteed to return the same solution every time they are queried. In the event that multiple minimal solutions to the optimization problem have equal cost, the QPU may return any of the solutions [75]. Secondly, because of the frequent optimization requests, the vehicles' projected positions often stayed the same between optimization requests, causing the same QUBO problem to be formulated in successive requests, which is not particularly useful. After testing various optimization intervals (both faster and slower), 120 seconds between optimization requests resolved the issue, allowing for sufficient time to change the projected locations of the vehicles.

5.3.2 Final tests: Lisbon

The second testing phase occurred during September 2019, with a small number of buses and drivers in Lisbon. The final bus route start and end points as selected in Section 5.1 were tested using the Android navigation app and the quantum web service, including the changes implemented after the initial testing phase in Wolfsburg. The significantly different conditions in Lisbon allowed us to further tune our navigation system, explained below.

Street exclusions. The road network of Lisbon is significantly different from that of Wolfsburg. Apart from the major roads and highways in Lisbon, many of the local streets are narrow and one-way, making them not well suited for public transit. Additionally, many roads have steep inclines, which are difficult for buses to climb. Neither of these two conditions were present in Wolfsburg, which lead to multiple unacceptable scenarios when testing the buses in Lisbon. More than once, routes were suggested that utilized these small streets through which the buses could not fit, causing them to either turn back (adding delay to the travel time), or even worse, forcing them to stop completely. For example, between Saldanha roundabout and Alameda station there is a network of highly connected one-way streets. Suggested routes using these roads could be useful for cars, but were completely undesirable for the buses in the Quantum Shuttle fleet. Similarly, a network of narrow one-way streets exists near the Web Summit conference center, which also needed to be avoided. To accomplish this, whenever routes were suggested that utilized roads in these networks, the GPS locations were recorded, added to a list stored on the QWS server. This list (in the form of bounding boxes of areas to avoid on a map) was submitted as part of the HERE routing API request, which returned only routes that avoided those areas. This list of forbidden areas was actively updated throughout the successive tests in Lisbon based on feedback from the bus drivers, resulting in only valid routes being generated by the end of the final testing phase. This list of excluded regions was then used for the live run during the Web Summit.

Time filtering. The number of candidate routes that can be requested from the HERE routing API per vehicle is an unconstrained parameter client-side. By default, 3 candidate routes per vehicle were requested, to keep consistent with the work in [98]. However, it was observed that in some cases one or more of the routes suggested were significantly slower than the fastest route suggested, resulting in extremely slow routes sometimes being suggested. The reason for these slow routes being selected was due to the way the cost functions are formulated in the QUBO problem. The goal is to reduce the amount of congestion caused by the vehicles, defined by the number of streets/GPS points shared between candidate routes across all vehicles. The slower routes suggested by the HERE routing API were often significantly longer than the fastest suggested route, and thus had lower overlap with the faster routes, causing some of the vehicles to be assigned to the slower routes. To circumvent this, a time filter was implemented to assure only reasonably fast routes were considered as valid candidates. After testing various values for the time filter, a value of 2 minutes provided the best trade-off between the number of routes selected and the routes' relative expected travel times. By allowing the slowest suggested route to be *at most* two minutes slower than the fastest route suggested, we were able to maintain three valid candidates per vehicle for the majority of the trips.

5.3.3 Web Summit 2019: Live run

For the launch of the event at the Web Summit conference, this project was presented under the name “Quantum Shuttle”. The service was active from November 4-7, 2019, and was operational for public use from November 5, 8:00 in the morning Lisbon time, to 18:00 in the evening on November 7, 2019. A total of 185 trips were recorded during the 4 day period with a total fleet size of 9 buses.

However, a small number of trips were erroneously recorded, due to manual driver cancellation or restart of the trip. Such trips were identified in two ways: either a small number of vehicle locations recorded in the database (fewer than 5), or one of the origin/destination points being far from the expected location (more than 1 km). Of the 185 trips, 162 (87.6%) were valid trips corresponding to the expected "Quantum Shuttle" service. The exact counts per day and line of the service are presented in Table 5.1. As per Section 5.1, two of the lines operated from the city center of Lisbon to the Web Summit: Alameda station to Web Summit (blue line) and Saldanha roundabout to the Web Summit (red line[1]). The third line (black) ran from the Web Summit to Alameda station.

	Black line	**Blue line**	**Red line**
Conference total	53	56	53
8:00-12:00	17	21	47
12:00-18:00	36	35	6

Table 5.1: The 162 trips taken by Carris buses operating the Quantum Shuttle, separated by time of day and line.

All three lines had roughly equal number of trips throughout the conference. The route frequency matches the expected demand in Lisbon– the red line from Saldanha was the popular choice during the morning, whereas the blue line from Alameda was used more in the afternoon. Likewise, the only line from the Web Summit back to Lisbon city center (black line) had double the trip frequency in the afternoon compared to the morning, again matching the demand of conference attendees returning to their accommodations after the conference ended each day. The duration of each trip was recorded together with the location history of each vehicle in the fleet throughout the conference. The corresponding average trip times are shown in Table 5.2. The trip times are recorded from the moment a driver presses the start button, until either the trip is manually ended or the bus is within 50 meters of its destination.

One of the key design goals in our traffic navigation system was making sure it could operate continually without manual intervention. As a consequence, there was significant variation in the complexity of the optimization problems being solved throughout the conference, depending on the number of active vehicles in

[1]Since the quantum navigation of the green and red lines have identical origin and destination, we combine them and refer to them together as the red line.

	Black line	**Blue line**	**Red line**
Conference total	23 min 41 s	23 min 18 s	26 min 34 s
8:00-12:00	25 min 36 s	22 min 43 s	27 min 36 s
12:00-18:00	22 min 46 s	23 min 38 s	18 min 19 s

Table 5.2: Average trip times for the Quantum Shuttle, separated by time of day and line.

the fleet. A total of 1275 optimization problems were solved by the QWS for the 162 trips, with an average response time of 4.69 seconds. Of those, 728 problems (57.1%) involved more than one route per vehicle in the system, with an average response time of 6.78 seconds. We consider the optimization problems for which there is more than one route per vehicle as the "harder" version of the traffic flow problem, since otherwise the route selection is trivial. It is important to note that the ability to navigate the fleet buses strongly depended on creating and solving the optimization problems in a timely manner. Since we cannot anticipate in advance whether the vehicles have one or more possible routes (depending on the traffic conditions), our system needed to operate uninterrupted in all cases. Additionally, while there was a fallback mechanism in place stored locally on each device, 100% of the calls to the D-Wave HSS completed successfully, thus maintaining our automated navigation system's integrity for the duration of the conference, regardless of the complexity of the problem being solved[1].

The goal of this project was to evaluate the use of solving the QUBO formulation of the traffic-flow optimization problem in a live setting to perform turn-by-turn navigation. Therefore, we now focus on interpreting the data and results of the project in this application context. The distribution of all the Quantum Shuttle trip data is shown on a map of Lisbon in Fig. 5.3. The trips are colored based on the lines to which they belong, as described above. The red, blue, and black circles correspond the the origins of their respective lines. The black circle is the Web Summit conference location, and is therefore also the destination for the red and blue lines. It is important to note that none of the three lines used the same route for all trips throughout the Web Summit, showing that our QWS navigation system provided flexible traffic-aware routing. The three highways that connect between

[1]The largest QUBO that was solved consisted of 12 variables, with 5 buses being navigated concurrently. This occurred on November 5, 9:11 Lisbon time, which was the busiest period during the conference.

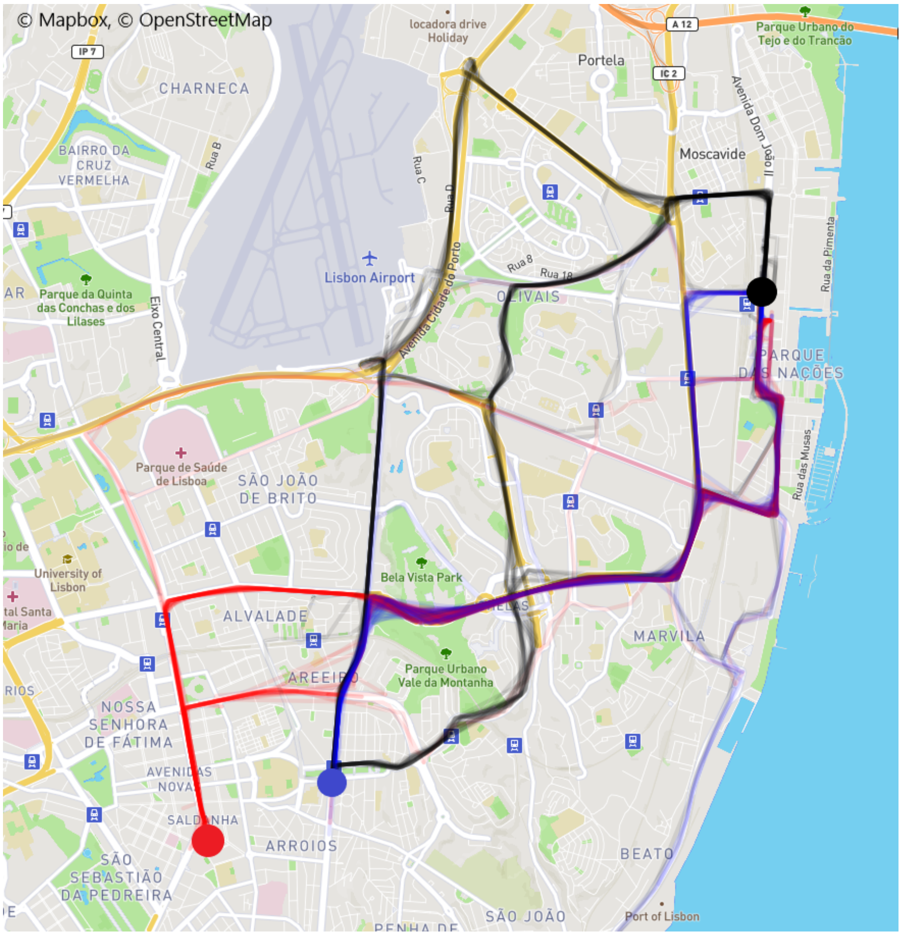

Figure 5.3: Distribution of all recorded Quantum Shuttle trips. The trips are color-coded based on the line they correspond to. The circles represent the origins and destinations of the respective lines [124].

the city center and the conference center were used extensively (although not exclusively), and at different times by the different lines. That these highways were prominent in the route selections is attributed to two design choices: time filtering and excluded streets. Highways are typically the fastest method of driving medium- and long-range distances, making them likely candidates for selection. Furthermore, the regions that were excluded from the route selection as per Section 5.3.2 removed fast route suggestions that avoided highways. It is reasonable to assume that in the case of navigating cars, as opposed to buses, Figure 5.3 would show increased distribution over the smaller city streets as well.

To quantify the customization of the routes used by the vehicles, we measure the dissimilarity between them. Specifically, we measure the overlap between the location histories of vehicles being navigated concurrently by our system. The overlap is defined as the fraction of GPS points in a vehicle's location history that coincided with another vehicle's route (that was being navigated at the same time), within a distance of 50 meters[1]. The overlap metric is therefore defined to lie between [0, 1]. Distance (d) between GPS points was calculated using the haversine formula:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos\phi_1 \cos\phi_2 \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right), \tag{5.3}$$

where ϕ_1, ϕ_2 are latitudes, λ_1, λ_2 are longitudes, and $r = 6,371,009$ meters is the Earth's radius.

Because the navigation system re-optimized the distribution of routes at most every 120 seconds, each vehicle's recorded trip is a sum of successive routes suggested by the optimization. It is important to note that the suggested routes, as obtained via the HERE API, are traffic-aware, and therefore already circumvent the existing traffic congestion in the city. Thus, by minimizing the overlap between the fleet vehicles, we minimized the additional traffic congestion caused by our fleet. In Figure 5.4 we show the overlap between pairs of buses, grouped by the lines the buses are following. There are six possible ways to compare the buses this way: three within the same line (red-red, blue-blue, black-black), and three between the lines (red-blue, red-black, and blue-black).

To contextualize the results in Figure 5.4, we also calculate the overlap between the most direct (i.e., fastest) routes suggested by the HERE API for each line without the traffic-aware component. This therefore simulates our navigation system in an "offline" mode– public buses typically have pre-defined routes that are not deviated from even in the presence of traffic congestion. Using these static routes, we obtain the following overlaps: 0.70 for red-blue, 0.01 for red-black, and 0.16 for blue-black (intra-line overlaps are trivially 1, since the same routes would be used for every vehicle in that line). The red-black and blue-black overlaps are similar in offline

[1]This definition differs from the one in Section 5.2.2, since the recorded location histories were irregular, as opposed to the points used to construct the traffic-flow optimization problem. The location histories were interpolated using the HERE API for consistency.

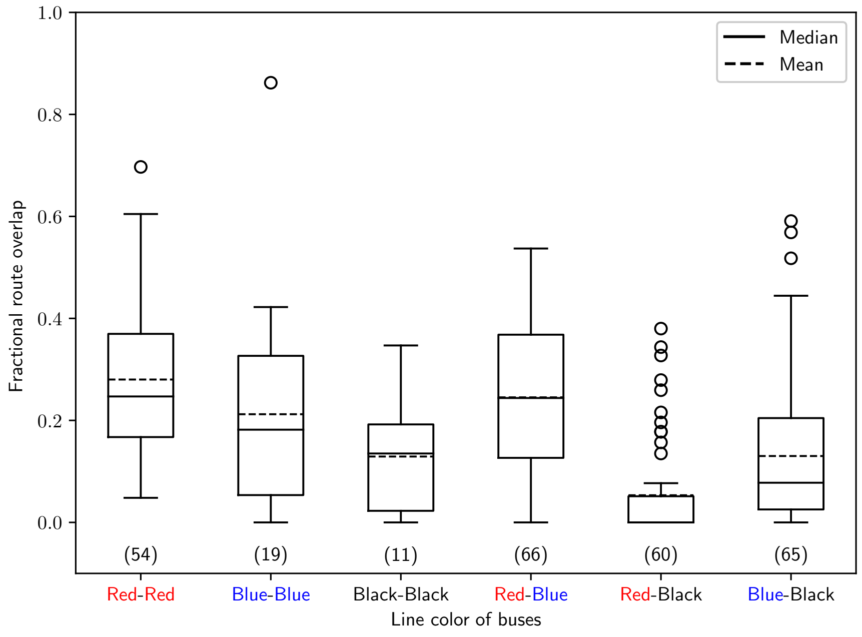

Figure 5.4: Box plot showing the fractional overlap of routes being navigated simultaneously in the Quantum Shuttle fleet. Boxes are grouped by the line colors. The number of observations are in parentheses below the boxes.

and online mode, however the expected time for the offline mode would be higher due to the lack of traffic-aware routing. For the red-blue overlap (as well as the intra-route overlaps), our online method significantly reduces the overlap between the routes. It is also worth noting that in all categories in Figure 5.4, the mean and median overlaps were below 0.5, meaning that the majority of every route was different from every other vehicle in the fleet. By using the QWS we were therefore able to both circumvent existing congestion as well as avoid creating new congestion.

From a technological perspective, the Quantum Web Service's modular implementation allowed us to communicate with a live QPU in a timely fashion, making it suitable for our traffic optimization use-case. The test runs in Wolfsburg and Lisbon were particularly instrumental, allowing us to fine-tune the connections between the components of the QWS given the constraints of the application. Due to this, the final implementation of the Quantum Web Service can handle live optimization of other, non-traffic related processes provided modifications to the QUBO construction and live data components. Many production processes have similar constraints to the those present in the Quantum Shuttle, making our work adaptable to other scenarios. However, we stress that the problem sizes investigated here were small (no more than 30 variables in all cases), well within the range of problem sizes addressable by classical algorithms. Therefore, while the application ran adequately in a live environment for a demonstration, further research is necessary to fully understand how well the problem can be solved by quantum annealing at scale.

5.4 Motivating a better real-world optimization use-case

We now explore another real-world optimization problem from the automotive industry. The *paint shop problem* refers to a set of combinatorial optimization problems where the objective is to color a (fixed) sequence of cars with a fixed number of colors such that the total number of color switches is minimized, given a set of customer orders. This simple problem poses interesting scientific questions, which in turn have real impact for solving such problems in practice. The paint shop problem was originally posed by Epping et. al [125] as a form of coloring problem. Some clarification on nomenclature: the given car sequence can also be referred to as a word, where each car is denoted by a character. In [125], it was shown that the paint shop problem is NP-complete in both the number of colors and cars in the sequence. Furthermore, results show that, for bounded numbers of colors and unique cars, there exists a polynomial-time dynamic programming solution to these instances. Subsequent work [126] extended these results, proving that even the simplest coloring version, with only two colors, is both NP-complete and APX-hard. Additional results show that a subset of problems meeting specific conditions can be solved in polynomial time. Therefore, this class of problems are good candidates for quantum optimization algorithms, and initial such studies have been conducted on a restricted definition of the problem [127] (different sub-classes are distinguished shortly).

5.4.1 The paint shop optimization problem

One of the steps in car production at Volkswagen is painting the car body before assembly. In general, this can be viewed as a queue of car bodies that enter the paint shop, undergo the painting procedure, and exit the paint shop. It is important to note that the area of the factory immediately proceeding the paint shop is typically assembly, where car components are assembled into the car bodies. Because of the many different models and configurations being produced, designing a sequence of cars to be assembled that is optimal (also known as the car sequencing problem) is a known NP-hard problem in itself [128]. In practice it is imperative to solve the sequencing problem because of worker safety and regulatory issues, and we therefore treat the sequence of cars entering the paint shop as a fixed queue. However, the *colors* assigned to the cars within a given sequence are still randomly distributed, and thus we can focus on optimizing them. An example with three different car groups is shown in Fig. 5.5 (top).

Each car body entering the paint shop is painted independently in two steps: the first layer is called the filler, which covers the car body with an initial coat of paint, and the final color layer is the base coat, which is painted on top of the filler. The base coat is the color that matches the final customer order: blue, green, etc. However, the filler has only two possible colors: white for the lighter base coats colors, and black for darker colors. We define a customer order as the number of cars of each configuration to be painted one of the color choices. We associate each coating step (filler or base coat) with a unique class of paint shop problems, each of which are NP-complete [125, 126]. A simple version of the filler optimization is when there are only two cars of each unique configuration in the sequence and each needs to be painted a different color; this is referred to as the *binary paint shop problem* (BPSP) [127]. More generally, the filler optimization is referred to as the *multi-car paint shop* problem, where the cardinality of each set of unique cars in the sequence is unconstrained, but the cardinality of the color set is restricted to two (e.g., black or white). The base coat optimization is therefore an extension of the MCPS problem, where the cardinality of the color set is also unconstrained– we call this version of the problem the *multi-car multi-color paint shop* problem. In our work we focus on the filler optimization, the MCPS problem, which can be formulated natively as a binary optimization problem. We formally define the problem as follows:

Given:	a word w defining the fixed sequence of N cars (w_i denotes the ith character in w),
	set of $C = \{C_1, \ldots, C_M\}$ unique configuration groups,
	binary choice of colors $\{W, B\}$,
	function $k(C_i)$ which defines the number of w_i to be painted B in w,
	function $f(w)$ to count the number of color switches in w,
such that:	$\#w_i\vert_B = k(C_i),\ \forall C_i \in C$,
minimize:	$f(w)$.

In practice (i.e., in the real paint shop), the information required to formulate this optimization problem is always available, as it is a necessary part of fulfilling customer orders. Therefore, this MCPS problem representation above corresponds exactly to the industrial use-case of paint shop optimization on the filler line. This provides a more tangible use-case to solve using the hybrid quantum optimization methods developed in the previous sections. In Fig. 5.5 we show a simple example of the MCPS problem with three car groups.

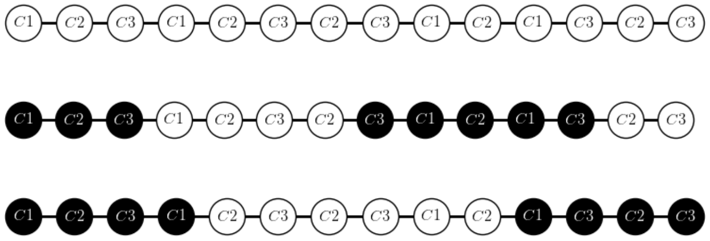
C1 C2 C3 C1 C2 C3 C2 C3 C1 C2 C1 C3 C2 C3
C1 C2 C3 C1 C2 C3 C2 C3 C1 C2 C1 C3 C2 C3
C1 C2 C3 C1 C2 C3 C2 C3 C1 C2 C1 C3 C2 C3

Figure 5.5: Simple example of a multi-car paint shop problem with three car groups (C_1, C_2, C_3). The three corresponding orders are $k(C_1) = 3, k(C_2) = 2$, and $k(C_3) = 3$. **Top:** The fixed sequence of cars in the paint shop queue. **Middle:** Sub-optimal solution to the problem with 3 color switches. **Bottom:** Optimal solution to the problem with 2 color switches.

Despite the focus of our work on the filler, we briefly address the optimization of the base coat. Although the filler and base coat are painted independently in separate locations, computationally the two problems are not separable. Abstractly, optimizing the base coat line (i.e., solving the multi-car multi-color paint shop problem) is a straightforward generalization of the MCPS problem: we can extend the binary color variables to discrete color variables, where $k(C_i)$ denotes the number of times each color appears in a car group. In practice, it is useful to consider solving the multi-color problem *after* solving the MCPS problem. Due to the aforementioned one-to-one mapping between base coat and filler color, it is still possible to permute the order of base coat colors within a contiguous sequence of filler colors to reduce the base coat color switches. Although discrete optimization problems can be represented as binary optimization problems, we do not solve the multi-color version of the problem and leave this work for future studies.

5.4.2 Ising model representation of MCPS

In principle, the formulation of the MCPS as an Ising model is straightforward: we start by representing every car in the sequence w (w_i) with a single spin variable (s_i). The spin up state denotes if the car is painted black, and the spin down state denotes if it is white. The Ising model which represents our problem can be divided into a hard constraint component and an optimization component. The optimization component is a simple Ising ferromagnet with $J = -1$ couplings between adjacent cars in the sequence:

$$H_A = -\sum_{i=0}^{N-2} s_i s_{i+1}. \tag{5.4}$$

This incentivizes adjacent cars to have the same color. The second component of our Ising model is the hard constraint, ensuring that the correct number of cars are colored white/black per customer orders. This is encoded in a second energy function, as a sum over independent k-hot constraint for each group of cars C_i:

$$H_B(C_i) = (\#C_i - 2k(C_i)) \sum_i s_i + \sum_{i<j} s_i s_j. \tag{5.5}$$

Therefore, the final Ising model is the sum of the two components:

$$H_{\mathrm{MCPS}} = -\sum_{i=0}^{N-2} s_i s_{i+1} + \lambda \sum_{C_i \in C} \left[(\#C_i - 2k(C_i)) \sum_i s_i + \sum_{i<j} s_i s_j \right], \quad (5.6)$$

with terms as previously defined in the MCPS problem statement. In order to ensure only valid configurations of spins are encoded in the ground state, it is necessary to scale H_B by the factor λ. The value of λ is chosen to be large enough such that it is never energetically favorable to violate a constraint to reduce the energy of the system. In our study we set $\lambda = N$, the total number of cars in our sequence, which guarantees this condition.

5.5 Creating Ising models from paint shop data

5.5.1 Data sources

The MCPS problem instances we used were generated from real data taken from a Volkswagen paint shop in Wolfsburg, Germany. The reason for this is two-fold: firstly, the main goal of this work is to test the viability of quantum annealing methods in solving industrial optimization problems. It is our goal to accurately capture the complexity of the industrial use-case without relying on simplifications or randomly-generated problem instances. Secondly, the paint shop currently operates on a first-come-first-served basis, where customer orders are entered into the queue as soon as they arrive. This guarantees a certain amount of randomness (although not uniformity) in the problem instances we solve: different car models, configurations, and base coat colors appear throughout the sequences in ways we do not control. Therefore, these conditions are suitable for our analysis.

A total of 104,334 cars were used in this study. To reproduce real-world conditions as faithfully as possible, the car sequences used are multiple independent sets of car sequences, each representing one week of continuous production, which are stitched together as one continuous block. The data is collected over a period of one year, roughly once every six weeks, to avoid seasonal biases in customer order preferences. There are 121 unique car configurations in the data set. Of those, 13 of the configurations appeared only once in the data set, and therefore do not need optimization at all. Typically this indicates that either a custom configuration was built that cannot normally be ordered, or a prototype assembled for testing purposes. Rather than exclude these from the study, we include them in the optimization, because fixing a color at one location in the sequence influences the adjacent cars (and consequently the total number of color switches) in a non-trivial way. We do, however, eliminate the spin variable from the Ising model by conditioning on the color.

5.5.2 MCPS problem sizes

To generate a variety of input sizes from the full data set, we partition the data into different sized sequences without permuting the car order. This is motivated by the amount of cars that need to be optimized for different purposes. For example, the paint shop used as a basis for this analysis has a queue capacity of roughly 300 cars. This is not the *total* capacity of the paint shop, but rather the maximum number of cars that can be physically inside the paint shop queue before they are painted. We consider this a rough lower bound on the problem size for industrially-relevant problem instances. An upper bound is more difficult to establish. From a theoretical point of view, there is merit in investigating the behavior of quantum systems in the infinite size limit, as in [127]. From an industrial perspective, car orders can be placed weeks to months in advance, which would yield problem sizes of $10^3 - 10^5$ variables. In reality, real-time and last-minute adjustments (due to manufacturing problems, supply chain issues, or imperfections in painting) can happen on a daily basis. We limit the analysis to problems of up to 3000 variables, which roughly corresponds to a few days worth of production. The data partitioning is performed by dividing the entire data set into equal chunks for each problem size N. Each partition is considered a candidate instance, yielding a total of $\lfloor \frac{104{,}344}{N_{\text{cars}}} \rfloor$ partitions per problem size. We then mine these partitions and select suitable partitions to generate problem instances from. This is due to the fact that there could be little frustration within some partitions. For example, it is possible that all cars of any one configuration (C_i) *all* need to be painted either black or white. While in general this is an accurate reflection of production, for experimental analyses this scenario is not useful. Therefore we deem a data partition to be a usable MCPS instance if the total number of non-fixed cars is at least 70% of the cars in the partition. Meaning that, in a 10-car data partition, at least 7 of the cars must have the freedom of being painted either color. We show the total number of partitions for the various problem sizes and how many partitions were valid problem instances in Table 5.3. For our experiments we randomly selected 50 valid instances to test at each N, except for the largest problem size of which we use all 34 valid instances.

Table 5.3: Problem sizes and number of problem instances generated from the data set partitions.

Problem size (cars)	**Num. partitions**	**Num. instances (% of partitions)**
10	10,433	172 (1.6%)
30	3,477	418 (12.0%)
100	1,043	756 (72.5%)
300	347	341 (98.3%)
1000	104	102 (98.1%)
3000	34	34 (100%)

5.5.3 Classical, quantum, and hybrid solvers

To evaluate the efficacy of quantum (and hybrid) algorithms in both the small-scale and industrially-relevant limit, we validate our methods using multiple algorithms. The goal of this analysis is to provide a fair but thorough comparison of results across different regimes of the MCPS problem, from small toy problems to large-scale instances, and represent real-world optimization conditions as closely as possible.

Random. Without optimization, we consider any assignment of colors to cars in a sequence where the orders are fulfilled to be a valid, but not necessarily optimal, solution to the problem. Thus, we can trivially generate random sets of valid solutions by uniformly assigning the color black to $k(C_i)$ cars for each car group C_i. While far from optimal, this solution to the MCPS problem may be preferable if other steps in the car manufacturing process are valued over the painting step. This was indeed the case for the data obtained from the paint shop in Wolfsburg, and thus random valid solutions serves as the baseline the competition algorithms are tasked with beating. For our study, we generate $2N_{\text{cars}}$ random valid solutions at each problem size to estimate the number of color switches that would occur naturally in the paint shop.

Black-first. This is the simplest algorithm that is used to solve the MCPS problem. Starting at the beginning of the sequence, we greedily assign the color black to every car until a white color *must* be assigned to the next car. In greedily obtained solutions the number of color switches grows linearly with the number of cars and is sub-optimal except for a minority of cases [126, 127]. Nonetheless, it serves as a good benchmark for more sophisticated optimization algorithms, as the improvement over random assignments grows with the problem size.
Simulated annealing. The simulated annealing metaheuristic used here is the same as those in the previous analyses in this thesis [99].
Tabu search. The Tabu metaheuristic here is also the same as in the previous chapters, and the same implementation is used [99].
D-Wave 2000Q. One of two different D-Wave QPUs used in this study, this QPU was the older-generation Chimera architecture with maximum degree 6. The processor used in this study had 2041 functional qubits.
D-Wave Advantage. The newest-generation QPU provided by D-Wave with a different topology and a significantly higher qubit count than its predecessor. The new topology, Pegasus, had a maximum degree of 15, and the QPU used in our study contained 5436 functional qubits. For further information regarding the D-Wave QPU topologies and the differences between them, we refer the reader to [64].
D-Wave Hybrid Solver. At large instance sizes (300 cars and higher) it was no longer possible to embed problems directly onto both D-Wave QPUs. Therefore we employed the hybrid quantum-classical algorithm used previously, the Hybrid Solver Service (HSS). As before, the QPU it uses cannot be programmed fully directly by the user, and thus we treat this algorithm as an optimization black-box with a single timeout parameter.

5.6 Benchmarking solvers in the industrial limit

We interpret our results relative to two different regimes: small-scale (10-100 cars) and industrial (300-3000 cars). Each solver used in these experiments was tuned in good-faith, but not necessarily optimally. Meaning, considerable effort was made to ensure solvers were being used to their strengths, but fully optimizing over all sets of hyperparameters for the solvers was deemed out of scope. We compared all solvers' performance in terms of their "improvement" over the random solver, defined as the difference in $f(w)$ between the best solution obtained by each solver and the random solver. This metric is representative of the real-world expected improvement using each of the solvers. In Table. 5.4 we report the median for each solver. We quote the median to be less susceptible to tails of the distribution. The results therefore represent the typical MCPS case at each problem size, rather than the expected value of each solver's overlap with the ground states. The solutions obtained from all solvers were post-processed (if needed) to ensure that the $k(C_i)$ constraint was satisfied in each problem. We show how often this occurred in Table 5.4 as well. We consider this necessary in order to interpret the solutions relative to the application, since it is trivial to reduce the number of color switches by ignoring the customer order constraint. In Fig. 5.6 we highlight the empirical scaling of our results in the industrial limit.

Solving problems directly with both QPUs was only possible for problem sizes 10-100. Embeddings were generated using the standard D-Wave embedding tool Python package, the same that used in previous experiments in this thesis [68]. The chain strengths required by each QPU was different depending on the length of the chains in the embeddings. We calculated the algebraic chain strength $\text{chain}_i = |h_i| + \sum_{j\in\text{adj}(i)} |J_{ij}|$ for every spin in the Ising model, and introduce an additional scaling parameter $s = [0.1, 0.2, \ldots, 1]$. Thus, the chain strength is defined as $s \cdot \max(\text{chain}_i)$, where s was optimized per problem size using a subset of the instances (10 per size), and $50 \cdot N$ samples per instance. Optimal s was defined as the value which yielded the highest frequency of valid solutions relative to the constraints of the MCPS problem in Eq. 5.6 (not chain breaks). We found that the D-Wave Advantage QPU had optimal $s = 0.3$, whereas the D-Wave 2000Q QPU had optimal $s = 0.45$. This is consistent with the fact that the Advantage QPU required shorter chains to embed the same Ising models as the 2000Q. Each QPU was then sampled for $500 \cdot N$ samples per problem size N, with annealing time $t_a = 1\mu s$. We used N spin-reversal transforms for each sample set, as it has been shown that there are diminishing returns between 100-1000 samples per transform [75].

We found that for the 10 car instances both QPUs matched the consensus best results between all solvers (median of $f(w) = 2$), and for the 30 car instances very near the best results ($f(w) = 5$ as opposed to SA's and HSS's 4). For the 100 car instances, with 50,000 samples per problem, the 2000Q QPU found valid solutions for 22/50 instances, as opposed to 37/50 for the Advantage QPU. From this we conclude that 50,000 samples is insufficient for the QPUs, but due to limited time availability we could not take more samples. We note that it was also possible to embed 47/50 of the 300 car instances onto the Advantage QPU. However, due to the poor performance on smaller sizes with limited resources, we did not evaluate those problems. Furthermore, due to the limited problem sizes that could be solved with the QPUs and the quality of results, we do not include these results in Fig. 5.6.

The Tabu solver was given $\lfloor N/3 \rfloor$ seconds per problem as its timeout parameter, and all other parameters were set to their default value. This solver struggled to find valid solutions past the 10 car instances. Due to the post-processing technique, which greedily corrected each sample to satisfy order $k(C_i)$, the Tabu results were essentially a worse version of the greedy algorithm at all problem sizes but the smallest.

Simulated annealing (SA) has many tunable hyperparameters: number of sweeps (N_{sweeps}), number of samples (N_{samples}), and (inverse) temperature (β) schedule. In our experiments we fixed the schedule to $\beta = [0.01, 10]$, interpolated geometrically using N_{sweeps}. We set $N_{\text{sweeps}} = 10 \cdot N$ and $N_{\text{samples}} = 20 \cdot N$, where N is the number of cars. The solutions obtained by SA (shown in Fig. 5.6) were consistently better than the greedy algorithm. Using the given parameters SA was able to provide valid solutions for 50/50, 49/50, and 44/50 for the 30, 50, and 100 car instances. However, the timescales necessary to obtain results were prohibitive from extending the experiments: 300 variable problems were terminated after running for 24 hours without returning a solution. We include the SA results in Fig. 5.6 due to their high quality at small sizes. Using a single-threaded SA implementation, run-time of the algorithm was on the order of seconds to minutes for the 10 and 30 car instances, and between 1-3 hours for the 100 car instances. We note that SA was the only solver which was allotted quadratically scaling computing resources: both sweeps and samples scaled with N. This is necessary for SA to be competitive, and exemplifies the trade-off between results quality and algorithmic run-time when using heuristics.

The D-Wave HSS was given equal time to the Tabu solver, given its only parameter is the timeout: $\lfloor N/3 \rfloor$ seconds per problem. The HSS was the only solver to consistently provide better solutions than the greedy algorithm for all problem sizes. The improvement continued to grow with increasing problem size, shown in Fig. 5.6. However, the gap between the HSS and the greedy algorithm shrank with increasing problem size, performing only slightly better than greedy for the 3000 car instances.

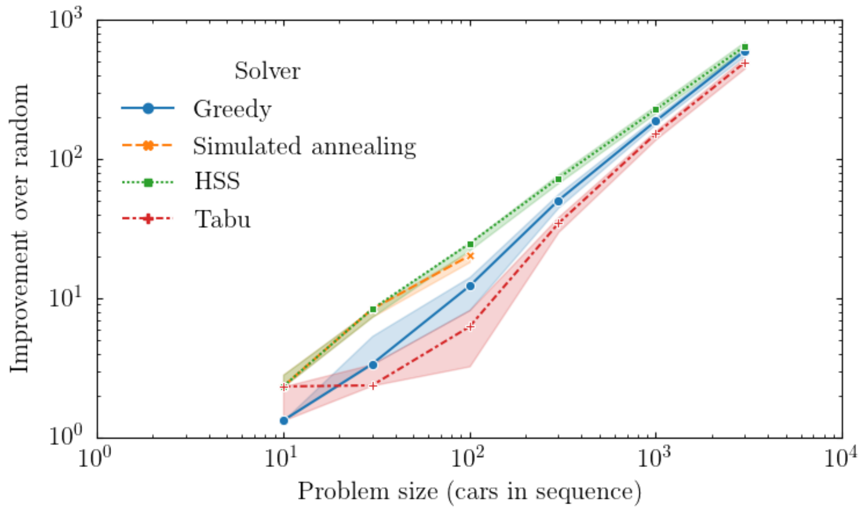

Figure 5.6: Number of color switches within the sequence shown as improvement over random configurations of orders.

Despite the simplicity of the Ising model representation of the MCPS problem, almost all solvers exhibited difficulty in finding valid solutions. This is particularly evident in the performance of the two different models of QPUs tested. The results degrading rapidly from 30 to 100 cars indicate that the problem became more difficult to solve disproportionately to the increase in system size. The SA and Tabu solvers exhibited similar trends, despite the increase in resources allotted to them. While the HSS was the best-performing algorithm, it also missed valid solutions for some problems of intermediate size (100 and 300 car instances). We identify two possible issues: first is the connectivity of the problem graph. Each order $k(C_i)$ requires a separate k-hot constraint which is represented using a fully-connected graph. This yields sub-cliques within each problem that increase with the problem size. In QPUs, denser problems create longer chains and higher chain strengths. For classical solvers, these sub-cliques create rugged landscapes which make single-flip optimization algorithms significantly less useful. Therefore, that the maximum sub-clique of the MCPS problem graph increases as a function of the number of cars is a bottleneck for performance. Secondly, the normalization terms necessary to encode the MCPS problem as an Ising model also scale with problem size. In Eq. 5.6, we set $\lambda = N$ to ensure the constraints are valid in the ground state of the Ising problem. Therefore, we observe that the numerical precision necessary to formulate the MCPS problem scales as $1/N$. This effect compresses the gaps between the local (and global) minima, making it harder to differentiate between them. Using a direct embedding approach for QPUs requires even higher precision due to chains: the chain strengths scale with N, and therefore the encoding precision as $1/N^2$. This may be prohibitively low in large-scale problems and analog devices due to finite control errors.

	Greedy		HSS		Tabu		SA		2000Q		Advantage		Random	
	$\%_{\text{valid}}$	$f(w)$	$\%_{\text{valid}}$	$f(w)$	$\%_{\text{valid}}$	$f(w)$	$\%_{\text{valid}}$	$f(w)$	$\%_{\text{valid}}$	$f(w)$	$\%_{\text{valid}}$	$f(w)$	$\%_{\text{valid}}$	$f(w)$
$N = 10$	100%	3	100%	2	86%	2	100%	2	100%	2	100%	2	100%	4.325
$N = 30$	100%	9	100%	4	26%	10	98%	4	100%	5	100%	5	100%	12.367
$N = 100$	100%	23	92%	10.5	0%	29	88%	15	44%	31	74%	28.5	100%	35.25
$N = 300$	100%	57	96%	34.5	0%	73	-	-	-	-	-	-	100%	107.58
$N = 1000$	100%	160	100%	121	0%	196	-	-	-	-	-	-	100%	348.35
$N = 3000$	100%	455	100%	406.5	0%	558.5	-	-	-	-	-	-	100%	1054.41

Table 5.4: Results for all solvers. We present the percentage of problems for which each solver found valid solutions ($\%_{\text{valid}}$) and the median number of color switches ($f(w)$) for each problem size N cars.