IEEE Transactions on Quantum Engineering

Received 21 January 2022; revised 29 July 2022; accepted 3 August 2022; date of publication 22 August 2022; date of current version 8 September 2022.

Digital Object Identifier 10.1109/TQE.2022.3199267

Deep Space Network Scheduling Using Quantum Annealing

ALEXANDRE GUILLAUME[1], EDWIN Y. GOH[1], MARK D. JOHNSTON[1], BRIAN D. WILSON[1], ANITA RAMANAN[2], FRANCES TIBBLE[2], AND BRAD LACKEY[2]

[1]Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109 USA
[2]Microsoft Quantum, Redmond, WA 98052 USA

Corresponding author: Alexandre Guillaume (e-mail: alexandre.guillaume@jpl.nasa.gov).

This work was supported by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (NASA).

ABSTRACT The National Aeronautics and Space Administration's (NASA) Deep Space Network (DSN) is responsible for communication and navigation for several NASA and international missions. The DSN comprises three complexes located in Goldstone (California, USA), Cambera (Australia), and Madrid (Spain). This distribution in longitude guarantees a full sky coverage. Each complex has one 70-m and several 34-m antennas. The network routinely serves a few dozen missions. The scheduling of the DSN is complex and involves human interventions as well as automated solutions. In order to increase the level of automation, different computing paradigms have been explored. In this article, we report on the quadratic unconstrained binary optimization (QUBO) formulation of the DSN scheduling, as well as a custom classical solver designed around some of the unique features of this scheduling problem. Thanks to a hybrid framework that extends the size of the problems that can be solved with a quantum annealer, we are able to generate a schedule from the QUBO formulation of the problem for one week's worth of user antenna requests, which represents the time period scheduled during operation. In other words, this work describes a real-world application of quantum annealing using real-world, operational data. We compare the resulting schedules' quality to solutions obtained using a mixed-integer linear programming formulation on a commercial solver. Our custom solver, based on a quantum-inspired optimization technique called substochatic Monte Carlo, while much faster in generating schedules, could only treat a subset of requests, and hence, we report its results independently.

INDEX TERMS Deep Space Network (DSN), quantum annealing, quantum inspired optimization, quadratic unconstrained binary optimization (QUBO), scheduling.

I. INTRODUCTION

The Deep Space Network (DSN) is the spacecraft tracking and communication infrastructure for the National Aeronautics and Space Administration's (NASA) deep space missions. It consists of three sites, approximately equally separated in (terrestrial) longitude, with multiple radio antennas at each site. Fig. 1 summarizes the key characteristics of the three sites.

In the course of a week, there are typically three dozen missions requesting access to one or more DSN antennas, with requests for time ranging from about one hour to several hours. These requests are for time to send commands to a spacecraft, receive data from a spacecraft, conduct observations of a spacecraft for navigational purposes, or even

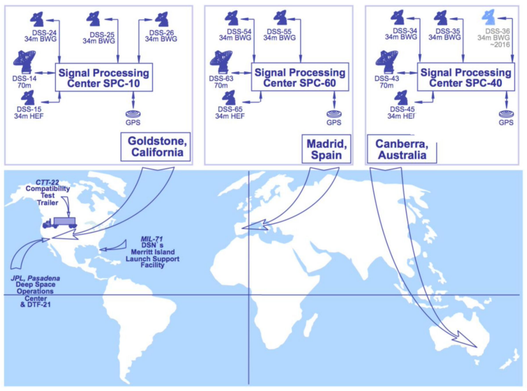

FIGURE 1. **DSN asset types and locations.**

conduct science observations of the spacecraft (radio or gravity science). Depending upon the nature of the request, only certain antennas may be able to conduct the track; for instance, because of its location in the sky, only antennas at the Canberra Deep Space Communications Complex (CDSCC) in Australia can transmit commands to or receive data from the Voyager 2 spacecraft and only a single antenna at the Goldstone DSCC can conduct certain gravity science observations with the Juno mission at Jupiter.

An integral aspect of DSN operations is to receive and organize the requests for access to the DSN antennas, then produce a valid schedule that accommodates the various requests subject to constraints of the missions. Further, there are operational constraints, such as antennas needing to be taken out of service for maintenance. Finally, though not frequent, there can be events associated with a mission that result in its requests being assigned a higher priority than those of other missions, a common example is when a spacecraft is entering orbit around an object in the solar system or landing on another body.

Because the DSN scheduling problem has been complex to solve with conventional technologies and this complexity is only projected to increase in the future, we undertook an effort to solve it using quantum and quantum-inspired computing technologies, which offer promising prospects for the future.

In summary, the contributions of this work are the following.

1) A QUBO formulation of the DSN scheduling.
2) Real-world problem results (obtained with D-Wave hybrid solver service) with more than a hundred million QUBO elements.
3) A quantum-inspired custom solver based on substochastic Monte Carlo.
4) A comparison of the results obtained with the QUBO formulation, mixed-integer linear programming (MILP) formulation, and quantum-inspired custom-solver.

A. DSN SCHEDULING

The DSN scheduling is complex process that can take up to a few months, with a typical lead time of four months. Several teams distributed around the world work with missions and users to determine their service needs and generate an initial draft schedule. Then, schedules are improved through an iterative process involving peer-to-peer negotiation between teams. The DSN scheduling process consists a three phases [1]–[5]. The *long-range planning and forecasting* period typically starts four to six months before execution. During this period, users' requirements are processed and different analysis (down time, future mission, ...) conducted. The *mid-range scheduling* phase is when detailed user requirements are specified, integrated, negotiated, and all tracking activities finalized in the schedule. Starting at roughly 4–5 months before execution, users specify their detailed scheduling requirements on a rolling weekly basis. The *near real-time* phase of DSN scheduling starts roughly 2–3 weeks from execution and includes the period through execution of all the scheduled activities.

The DSN scheduling software systems is heterogeneous and each phase of the process is handled by different tools [1]. The Service Scheduling Software (S^3) assembly is focused on the mid-range schedule and near real-time. This service is decentralized and a peer-to-peer collaborative environment. It also automates the scheduling of DSN activities with a request-driven approach. The DSN scheduling engine (DSE) is one of the central components of the S^3 system. The DSE has several functions: schedule conflict checking, schedule request interpretation, schedule repair, schedule query, and schedule optimization. The DSE is based on Automated Scheduling and Planning ENvironment (ASPEN) developed at JPL [6]. ASPEN takes a local, heuristic, iterative search approach to planning, scheduling, and optimization. This approach allows quick iterative improvements of the schedules. A drawback of using a local search is that one cannot guarantee that the candidate schedules are the best.

There have been several efforts in the past to increase the level of DSN scheduling automation in general, and explore nonlocal search algorithms in particular. In 1993, Bell [7] used a Lagrangian relaxation approach with an integer linear programming (ILP) formulation. Johnston [8] described a multiobjective scheduling technique implemented with a differential evolution (DE) algorithm. A parallel implementation of the same algorithm was explored later by Brown and Johnston [9]. A similar evolutionary computation technique known as genetic algorithm (GA) was used by Guillaume et al. [10]. More recently, a squeaky wheel optimization was used to study prioritization and oversubscribed scheduling of the DSN [3]. Recently, a deep reinforcement learning (RL) approach demonstrated that an agent could learn to outperform a random baseline through trial and error in a simulated DSN scheduling environment [11].

B. QUANTUM ANNEALING

Solving the DSN scheduling problem consists in placing as many tracks as possible within a finite time span (one week) while utilizing a limited amount of resources (antennas and equipment) and satisfying other constraints. Alternatively, it could be said that forming a schedule consists in choosing one combination of track placements among all possible combinations of track placements that satisfy the constraints. In this sense, DSN scheduling is a combinatorial problem.

Quantum annealing (QA) is a metaheuristic based on quantum mechanics for solving combinatorial optimization problems. The principle of QA is to encode the solution of a computational problem in the ground state of quantum Hamiltonian. The computation starts from an easy to prepare Hamiltonian and is evolved to a final problem Hamiltonian whose ground state ground state encodes the solution of the computational problem. In physics, such a problem Hamiltonian is known as the Ising model and is expressed in terms of spins up and down. Instead of half-integer variables representing the spins, the problem Hamiltonian can be expressed in terms of binary variable, $x \in \{0, 1\}$

$$H_P = \sum_i Q_{i,i} x_i + \sum_{i<j} Q_{i,j} x_i x_j. \tag{1}$$

The development of quantum annealing is motivated by the possibility that quantum fluctuations and tunneling could provide an advantage over conventional computing. For the past few years, the company D-Wave Inc. has manufactured quantum annealers that are commercially available and can be programmed by setting the linear and quadratic coefficients, $Q_{i,i}$ and $Q_{i,j}$, to study a particular problem Hamiltonian H_P [12]. This company has been using superconducting integrated-circuit technology to fabricate chips that contain approximately 5600 qubits for the Advantage chip released late September 2020. A qubit is a quantum bit. This elementary unit of information behaves quantum mechanically. It can be in a superposition of states and be quantum correlated to other qubits. Those circuits have a planar design where qubits are arranged in a regular array pattern. In practice, geometrical and physical considerations constrain the layout in a 2-D plane. Advantage's hardware graph topology is called Pegasus and is characterized by 15 connections to other qubits per qubits [13]. In other words, for a given qubit i there are 15 possible j's with potentially nonzero values $Q_{i,j}$.

Problems that can be solved by optimizing (1) are called *quadratic unconstrained binary optimization*, or QUBO, problems. It is worth noting that this formulation predates the advent of QA. However, it is probably the progress in quantum annealing technology and research that has spurred the research in alternative technologies to solve QUBO problems. Different approaches have been proposed and demonstrated to find solutions to Ising problems with artificial spins implemented with various physical systems: coherent spin machine [14], trapped ions [15], electromechanical system [16]. There are a few commercial options that can solve QUBO problems. Toshiba proposes a solution called simulated bifurcation machine (SBM) which is based on a numerical simulation of the adiabatic evolution of classical nonlinear Hamiltonian systems exhibiting bifurcation phenomena [17]. The maximum size of the Ising problem that can be solved corresponds to a 10 000 variables ($Q_{i,i}$'s) and/or 1 million finite ($Q_{i,j}$) elements [18]. Fujitsu has developed a digital annealer (DA) which is a specialized digital circuit that has 8192 bits fully connected. If we assume that the matrix Q in (1) is upper-triangular, then 8192 bits fully connected would correspond to approximately 33 million elements in the Q matrix. The D-Wave company launched its Hybrid Solver Service (HSS) in February 2020. This service bundles classical computation and its QA chips into one service to extend the size of the problems it can solve. HSS was upgraded in September 2020 when the larger size QA chip Advantage was released. This latest version can handle problems containing up to one million variables and two hundred million weights (elements of the matrix Q) [19]. For a fully connected problems, the number of variables the HSS can handle is 20,000.

The possibility to use QA to solve planning of deep space missions was mentioned as early as 2012 [20]. Venturelli et al. [21] described a job shop scheduling solver based on quantum annealing. The authors used D-Wave's system Vesuvius, also known as D-Wave Two, which had 509 qubits. The DSN scheduling we describe in this work is similar to the job shop scheduling problem (JSP) described in [21] in that it reduces scheduling to a decision problem which attempts to determine whether jobs (DSN requests) can all be scheduled within a prescribed time (of seven days). However, the real JSP scheduling problem is an optimization problem whose objective is usually to schedule all jobs (and operations) while minimizing the total makespan. Venturelli et al. consider the required modifications to their decision version of the JSP in order to solve the optimization version of the JSP. As we have mentioned above, the DSN scheduling is done on a rolling weekly basis, i.e., for a fixed period of seven days. For this reason, we are solving a decision problem and do not need to extend our model to solve a problem similar to the optimization version of the JSP.

In this article, we describe a QUBO formulation of the deep space network and the results obtained with D-Wave's HSS accessed via its leap cloud service.

C. INPUT DESCRIPTION

DSN users represent their needs to the S^3 software system as scheduling requests [1], [2], [22], [23]. Each such request is interpreted by the DSN scheduling engine. The main elements of a scheduling request are service specification, timing constraints, track relationships, priority, preferences, repetitions, and nonlocal time line constraints.

A set of DSN requests for a given week is generated using information contained in user loading profiles (ULPs), which are outputs of the loading analysis and planning software (LAPS) responsible for long-term planning. For each mission, ULPs specify the following:

1) the number of requested tracks;
2) the set of valid antennas for these tracks;
3) the requested duration for each track;
4) the minimum required duration for each track;
5) the required set up and tear down times for each track.

These ULPs are combined with view periods that are generated using ephemeris data downloaded from JPL's service preparation subsystem (SPS). A view period is a period a spacecraft is visible from a ground station. In addition to visibility constraints, the set of valid view periods for a given request also encodes the constraint on overlapping visibilities for arrayed requests that simultaneously utilize multiple antennas. View periods that overlap with maintenance periods are trimmed or discarded as necessary.

In this work, a DSN schedule request is characterized by a mission name, week number, year, duration, minimum duration, list of resources, track identity number, set up time, tear down time, and a list of view periods for each resource. Different activity setup and tear down times must be scheduled before and after tracking, respectively. A resource is indicated by the acronym DSS (for deep space station) followed by a number, e.g., DSS-14 for Goldstone 70 m antenna.

A slightly different version of week 40 of 2018 data was published as part of a larger set of data, named SatNet, that was created to allow benchmarking of satellite scheduling optimization schemes based on historical data from the NASA Deep Space Network. SatNet data are an anonymized version of the data where a mission's name is replaced by an identification number. Other than that, the overall data format, field names and contents are similar to the data used in this work. A description of SatNet data and a link to the GitHub repository where it can be downloaded can be found in [25].

II. QUBO FORMULATIONS

A schedule is generated after processing N requests $\mathcal{RQ} = \{RQ_1, \ldots, RQ_N\}$. A given request RQ_n can utilize any of the M resources (antennas or equipment) prescribed for this request $\mathcal{RS} = \{RS_1, \ldots, RS_M\}$. In turn, periods of visibility of a spacecraft from a particular ground station are calculated and define a set of K viewperiods $\mathcal{VP} = \{VP_1, \ldots, VP_K\}$ for each resource RS_m. The times when a spacecraft becomes visible or invisible, respectively, the *rise* and *set* times, rt and st, define the outer boundaries of a viewperiod. The transmission to the spacecraft can begin (end) slightly later (earlier) than the rise and set time (respectively). These transmission on and off times, tn and tf, delimit the inner boundaries within which a *track* can be scheduled. For each request, a *set up* and *tear down time* period, su and td, are given to prepare and remove (respectively) the equipment requested. Those times can occur anywhere between the rise and set times. In addition, a track duration d_r is prescribed for each request. An *activity* is defined as the period of time equal to the sum of the set up, track and tear down times. It starts at the start of the set up time and ends at the end of the tear down time. In this work, a track is an individual communication pass [23]. It is the period of time during which an antenna can track a spacecraft and communicate with it.

The objective of the DSN scheduling is to fulfill every request or, in other words, schedule precisely one activity per request. We adopt the following notation where the binary variable $x_{n,m,k,t}$ is equal to 1 when a track starts at time t with t within the viewperiod k calculated for resource m of request n, and 0 otherwise. With this notation, the objective function can be written

$$h_1(\bar{x}) = \sum_{n=1}^{N} \left(\left(\sum_{m=1}^{M} \sum_{k=1}^{K} x_{n,m,k,t} \right) - 1 \right)^2 \tag{2}$$

with $tn \leq t \leq tf - d_r$. As a reminder, tn and tf delimit the inner boundaries within which a track can be scheduled and d_r is a track duration prescribed for each request.

In addition, the schedule should be devoid of conflicts (as illustrated in Fig. 2) between requests that use the same resource at the same time. To this end, we seek to minimize the following constraint:

$$h_2(\bar{x}) = \sum_{(n,m,k,t)} \sum_{(n,'m,'k,'t')} x_{n,m,k,t} \quad x_{n,'m,'k,'t'} \tag{3}$$

with $n \neq n'$, $RS_m = RS_{m'}$, $t - su \leq t' - su' \leq t + d_r + td$ or $t' - su' \leq t - su \leq t' + d'_r + td'$. The product in the double sum is equal to 1 when two activities overlap in time and use common resources and 0 otherwise.

The overall cost function compounds both the objective and constraint

$$H = \alpha h_1(\bar{x}) + \beta h_2(\bar{x}). \tag{4}$$

Every request has a minimum duration d_{min} listed in its attributes. If it is different from the requested duration d_r, then the track can be shortened. In this case, the scheduled duration d is allowed to vary between d_{min} and d_r, $d_{min} \leq d \leq d_r$, and the above formulation is modified by replacing d_r by d and $x_{n,m,k,t}$ by $x_{n,m,k,t,d}$. Finally, we note that the time t in the above formulation needs to be discretized in practice. We discuss the practical implications of this process in Section III.

A. MULTIPLE ANTENNAS

Some missions require multiple antennas to be used at the *same* time. For instance, the Voyager 1 and 2 missions left the solar system and consequently several antennas are needed to amplify the communication signals on Earth. The above formulation can accommodate this additional constraint simply by changing the resource condition $RS_m = RS_{m'}$ for single antennas by $RS_m \cap RS_{m'} \neq \emptyset$ for multiple antennas in the conflict check (3), where at least one the resource set contains two or more antennas. In this case, the binary variable x indicates the scheduling of two tracks simultaneously in different antennas.

B. SPLIT TRACKS

Some requests require a track time that is substantially higher than most requests and therefore might be difficult to satisfy. It might be advantageous to split those requests into two or more segments. This advantage, however, is counterbalanced by the extra setup and tear down times necessary to schedule the additional track(s). So, even if in practice a track can be split into more than two subtracks, the penalty incurred in this case would be higher than the penalty for a split in two subtracks (due to the extra setup and tear down times). For this reason and simplicity sake, we allow splitting into two subtracks only in this work to demonstrate the ability of our QUBO formulation. Requests with a duration longer than 8 hours form the set of splittable requests $\mathcal{SRQ} = \{RQ_1, \ldots, RQ_{N_s}\}$. Each of those requests can be split into

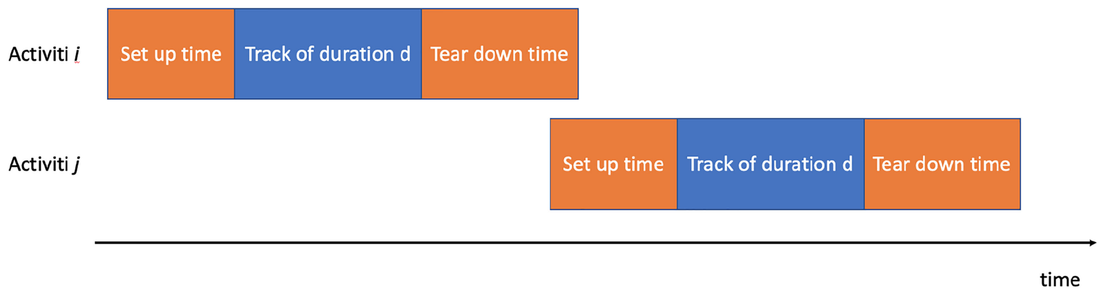

FIGURE 2. **Conflict between two activities *i* and *j*. A conflict exists if there is an overlap between two activities using the same resource. An activity is comprised of a set-up period, followed by a tracking period and a tear down period.**

two tracks, not allowing the shortest track to be shorter than 4 h. Similarly to the multiple antennas case above, the binary variable x indicates whether two tracks are scheduled.

III. RESULTS

All the results in this section were obtained with D-Wave leap hybrid solver service (HSS) using the Advantage chip. We used the hybrid solver `hybrid_binary_quadratic_model_version2` which was using the `Advantage_system1.1` solver as the default back end (that handled the quantum annealer at the time of the experiments, in July, August 2021). All programs are coded in Python. The process to generate schedules from DSN information using the HSS consists of three stages. First, a request list is read in and transformed in a QUBO matrix, Q see (1). Second, this matrix is submitted to the HSS, which subsequently returns results after a prescribed amount of time. Third, each binary string returned as a solution by the HSS is transformed into a schedule.

The first stage is itself comprised of two successive steps. The DSN information, or request list, is transformed into a list of binary variables $\{x\}$ and those variables are then used to calculate the cost function (4) or, equivalently, used to calculate the QUBO matrix. Since the first step is central to this work, we detail the transformation of a request list to a list of binary variables with the pseudocode Algorithm 1.

In this work, we are considering two sets of DSN requests (inputs), one for week 44 of 2016 and one for week 40 of 2018, to test our program and QUBO formulation. We chose week 40 of 2018 because it was the one of the most challenging to schedule as the difference between the time requested and the time actually scheduled was the largest during the period starting the first week of 2018 and week 21 of 2019 [24]. The main characteristics of these two datasets are indicated in Table 1. Both datasets offer the flexibility to schedule shorter than requested tracks, with 37.3% (47.7%) of their requests allowing shortening for week 44 of 2016 (week 40 of 2018, respectively). Another feature that facilitates scheduling is the ability to split long requests. There are relatively less requests than can be split than shortened with 10.9% of week 44, 2016 requests and 4.2% of week 40, 2018 requests. Antenna arrays, on the other hands, are more difficult to schedule because one request requires the use of several antennas. Fortunately, there are relatively few of those with 3.9% (7.5%) of week 44, 2016 (week 40, 2018) requests.

Numbers characterizing the first stage of schedule generation, the QUBO generation process, are indicated in Table 2 below. It is important to note that the QUBO generation process is an *embarrassingly parallel* problem, i.e., a computing problem that can easily be split into an arbitrary number of smaller independent tasks than can be solved in parallel. For a number of variables N_{var}, this first step is $O(N_{var}^2)$ in both classical time and memory. Thanks to this polynomial complexity and the embarrassingly parallel nature of this stage, the run time for this stage could be made as small as needed by parallelizing the code.

The run time of the second stage of the computation, the resolution of the QUBO problem itself, was set to 30 min for both datasets, by setting the hybrid solver option `time_limit` to 1800 s. The choice of this run time is arbitrary and was motivated by practical considerations. First, there exists a lower bound `min_time_limit` for the run time of leap hybrid sampler that depends on the number of QUBO elements, which was equal to 369 and 403 s for week 44, 2016 and week 40, 2018 datasets, respectively. Second, HSS computation is charged proportionally to the run time. This fact motivates the user to strike a compromise between the number of runs and their performances, setting a practical upper bound on the run time. Half hour run time was a good compromise for this work. The hybrid solver returned the amount of time used by the quantum processing unit (QPU), 390,070 microseconds and 519,754 microseconds for week 44, 2016 and week 40, 2018, respectively. These represent 0.02% and 0.03% of the total computation time (1800 seconds). Table 3 below summarizes the main characteristics of the solutions (schedules) obtained with the 2 datasets. 86.3% (80.8%) of all requests of week 44 of 2016 (week 40 of 2018) were satisfied. Both schedules do not contain any conflicts.

One of the two schedules, for week 40 of 2018, is represented in Fig. 3. Different activities are represented with different rectangles on this Gantt chart. An activity is the period of time starting at the start of the set-up period and ending at the end of the tear-down period that immediately follows the track itself. The maintenance periods, in black in Fig. 3, were added after the schedule was generated. In

Algorithm 1: Binary Variables Generation from DSN Data.

```
input  : A request list ${requests}$
output: A binary variable list ${x}$
// Generate 3 lists of candidate start times ${cst}$, ${cst_1}$ and ${cst_2}$:
for request in ${requests}$ do
    // Iterate over durations in 30-minute increments:
    for $d$ in ${d_{min}, d_{min}+30, d_{min}+60, ..., d_r - d_{min}}$ do
        // Generate candidate start time list ${cst}$ for single tracks:
        for resource in request resources do
            for view-period in resource view-periods do
                // Define a candidate start time $t$ every 15 minutes of the view-period:
                $cst$[request][resource][view-period][i] $\leftarrow (t, d)$
            end
        end
        // Generate candidate start time lists ${cst_1}$ and ${cst_2}$ for split tracks
        // (single request satisfied across 2 tracks):
        if $d_r > 8$ hours then
            for $d_1 \leftarrow d_{min}$ to $d_r - d_{min}$ do
                $d_2 \leftarrow d - d_1$ for resources$_1$ in request do
                    for view-period$_1$ in resource$_1$ view-periods do
                        // Generate a candidate start time $t_1$ every 15 minutes ($track_1$):
                        $cst_1$[request][resource$_1$][view-period$_1$][i] $\leftarrow (t_1, d_1)$
                        for resources$_2$ in request do
                            for view-period$_2$ in resource$_2$ view-periods do
                                // Generate a candidate start time $t_2$ every 15 minutes ($track_2$):
                                $cst_2$[request][resource$_2$][view-period$_2$][j] $\leftarrow (t_2, d_2)$
                            end
                        end
                    end
                end
            end
        end
    end
end
// Create binary variable list ${x}$:
for each candidate start times $i, j$ in ${cst_1}, {cst_2}$ do
    $track_1 \leftarrow (request_1, resource_1, view\ period_1, i)$
    $track_2 \leftarrow (request_2, resource_2, view\ period_2, j)$
    // Binary variable for a split track:
    $x \leftarrow (track_1, track_2)$
end
Repeat for single tracks list ${cst}$ gives $x \leftarrow (track, track)$.
```

other words, the maintenance periods were not part of the optimization process.

In order to illustrate the ability of the QUBO formulation to handle the splitting of long tracks, we have represented in Fig. 4 the four tracks resulting from splitting one Juno (JNO) mission request and one New Horizons Pluto mission (NHPC) request.

Multiple-antenna requests are also well handled by the QUBO formulation as illustrated in Fig. 5 with only the multiantenna tracks displayed.

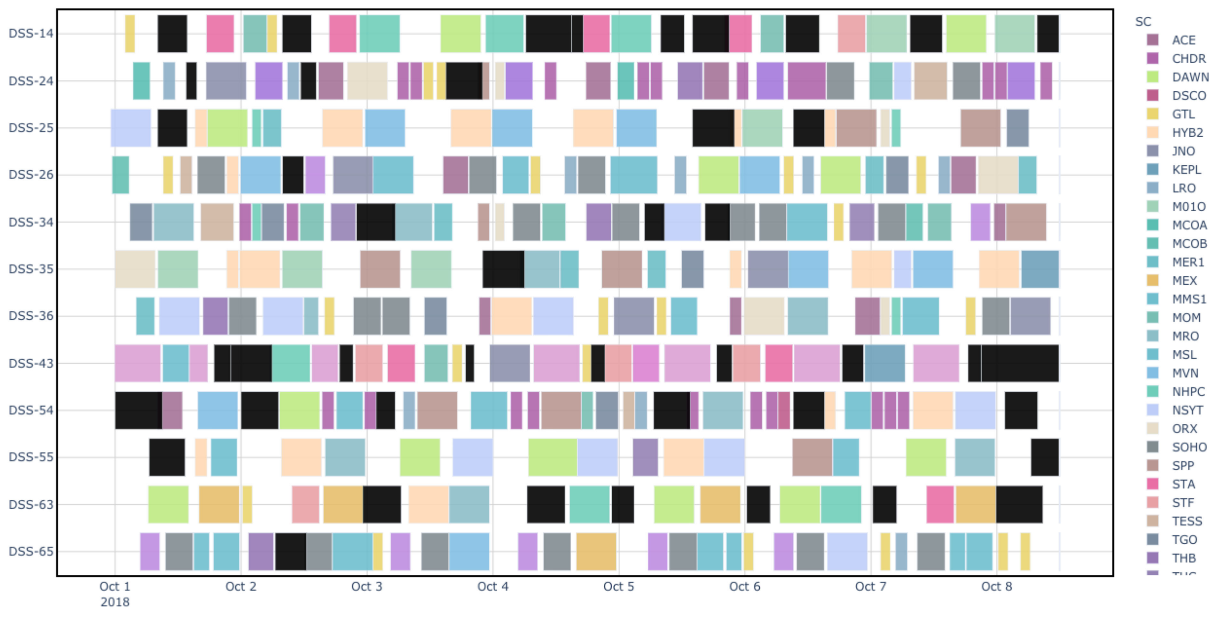

FIGURE 3. Week 40 of 2018 schedule generated with a quantum-classical solver. Each activity is represented by a colored rectangle. The abscissa spans one week period. The different antennas used are listed on the ordinate axis. Each antennas is labeled with the acronym DSS (for deep space station) followed by a unique identification number. The geolocation and type of each of those antennas is shown in Fig. 1. Different missions are indicated by different colors on the right legend. Maintenance periods are represented in black. The name of each mission corresponding to an acronym can be found on the Current Mission Set web page at https://deepspace.jpl.nasa.gov/about/commitments-office/current-mission-set/.

TABLE 1. Request Lists (Inputs) Characteristics

	Number	$d_{\min} < d$	Array	Duration > 8h
week 44, 2016	284	106	11	31
week 40, 2018	333	159	25	14

The second column indicates the number of requests that can be shortened from a duration d to $d_{\min}$. The third column indicates the number of requests that ask for multiple antennas. The last column indicates the number of requests that can result in several smaller tracks, i.e. that can be split.

TABLE 2. QUBO Characteristics

	Variables	Elements	Generation time
week 44, 2016	59,604	47,050,760	1:25:25
week 40, 2018	65,565	36,530,592	1:43:55

The first column indicates the number of variables or the number of diagonal elements $Q_{i,i}$of the matrix Q. The number of finite elements $Q_{i,j}$ of the upper triangular matrix Q are listed in the second column. QUBO matrix Q generation time (h:min:s) is indicated in the third column.

TABLE 3. Schedules (Outputs) Characteristics

	Satisfied requests	Shortened tracks	Tracks	Multiple antenna	Split
Week 44, 2016	245	59	252	5	2
Week 40, 2018	269	93	279	10	0

More tracks are scheduled than requested because some requests ask for multiple-antenna and some long tracks can be split into several if it improves the schedule.

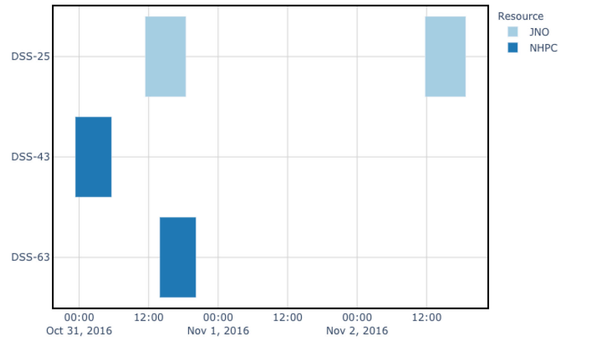

FIGURE 4. Week 44 of 2016 split requests.

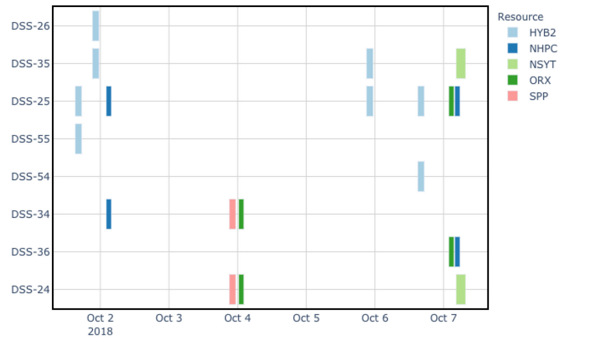

FIGURE 5. Week 40 of 2018 scheduled multiple-antenna requests.

As mentioned above, there are three successive stages to generate schedule out of DSN input information (weekly request list). As a consequence, the performance of the whole process depends on the performance of each stage.

The QUBO matrix Q generation process starts with a discretization of time within each view periods of a request list. This step associates a candidate start time and duration

TABLE 4. Comparison Between Quantum Hybrid (This Work) and MILP [26] Solvers

		# requests satisfied	tracks	scheduled hours	U_{rms} %	U_{max} %
2016	QUBO	245	252	975.90	31.5	89.2
	Δ-MILP(0)	216	260	900.00	35.6	86.4
	Δ-MILP(0)-L	215	266	923.25	38.0	100.0
2018	QUBO	269	279	1059.40	33.5	94.6
	Δ-MILP(0)	223	276	885.17	44.1	100.0
	Δ-MILP(0)-L	226	292	956.75	41.6	100.0

Results comparison between this work's and those obtained with a Mixed Integer Linear Programming algorithm [26] using the same DSN data. The run times for Δ-MILP(0) and Δ-MILP(0)-L were 30 and 450 minutes respectively. Year 2016 (2018) in the left column refers to week 44 of 2016 (week 40 of 2018, respectively).

TABLE 5. Custom Solver Solutions Characteristics

	Satisfied requests	Shortened tracks	Tracks	Split requests	Scheduled hours	Run time [s]
2016	242	63	259	17	1036.89	196
2018	261	94	273	12	1139.76	233

More tracks are scheduled than requested because some long tracks can be split into several if it improves the schedule.

to each request, antenna for this request and view-period for this chosen antenna. In this work, we used a 15 min time granularity between successive candidate start times. Then, a map relating this information to each binary variable is created. Each binary variable points to 2 lists of 4 quantities: a request index, an antenna (or list of antennas for multiple-antenna requests), a view-period index and a candidate start time index. This encoding allows to schedule one track as well as two tracks. If the two lists of four quantities for a given binary variable are either the same or different, one or two tracks can be scheduled, respectively. Finally, the QUBO matrix Q is evaluated according to (2)–(4) and the related information in Section II. The hyper-parameters α and β in (4) were set to 1 and 1.17, respectively.

In order to assess the performance of our QUBO formulation, we compare our results with those of Claudet et al. [26] who used the same data as we did with a conventional MILP algorithm. The comparison is summarized in Table 4. Claudet et al. ran their algorithm, called Δ-MILP(0), for 30 min and also provided us with results for 7.5 h runs indicated as Δ-MILP(0)-L in Table 4. The metrics U_{rms} and U_{max} are two measures of user satisfaction and are defined in [24]. The first is the root mean squared unsatisfied time fraction and the second, the maximum user unsatisfied time fraction. We give their mathematical expression in the next section with (6) and (7). The best (worst) user satisfaction would result in both those quantities being equal to 0 (1, respectively). The QUBO formulation schedules substantially more requests than the MILP approach for both datasets and run times. The unsatisfied time fraction is substantially higher for the long run of MILP than for the 30 min run indicating that extra requests were scheduled by sacrificing the overall user satisfaction. We notice that the MILP algorithm creates relatively more tracks compared to the number of requests satisfied than the QUBO formulation does. This means that although the MILP-based algorithm can split tracks into more tracks than the QUBO formulation can (two tracks), this ability does not result in a better user satisfaction. Our implementation of the QUBO formulation schedules more tracks, in less time, while achieving a better user satisfaction than the Δ-MILP(0) implementation of [26].

IV. CUSTOM SOLVER WITH RESULTS

We developed a custom solver specifically designed for the DSN scheduling problem. Owing to the limited time available for its development, we chose only to include request shortening and splitting; this version of the custom solver was not able to schedule multiple antenna requests. Consequently, runs with our implementation of the QUBO formulation result in schedules with more requests satisfied than runs with the custom solvers. The solutions generated with QUBO algorithm satisfied 245 and 269 requests while the solutions generated with the custom solver had satisfied 242 and 261 requests (see Table 5) for week 44, 2016 and week 40, 2018, respectively. Nonetheless the custom solver produces schedules with higher antenna utilization than either the QUBO or Δ-MILP generated solutions.

Although the custom solver is a classical solver, the principle behind the custom solver is quantum-inspired. The underlying optimization process is based on substochastic Monte Carlo [27], a diffusion Monte Carlo algorithm that arises from performing imaginary-time quantum adiabatic evolution. As with usual formulations of adiabatic evolution, one slowly evolves from a "driving Hamiltonian" that represents the kinetic energy of the search algorithm to a "problem Hamiltonian" that encodes the objective function as a diagonal potential energy operator. Implementing this on quantum hardware, that is to do quantum annealing, requires these Hamiltonians be engineered from natural processes. However emulating this in software allows for significantly more flexibility in our selection of kinetic and potential energy. For example, just as in the QUBO version, we opted to maximize the number of requests fulfilled (scaled by the relative amount a requestors time was shorten, when this occurs). However, other metrics could have been used with minimal modification to the solver.

The key advantage of the custom solver over a QUBO formalism is its ability to avoid penalty models. We built the dynamics of the custom solver so that its state always encodes a feasible schedule. Recall that in QUBO expressions of the scheduling problem, the state space consists of binary variables $x_{n,m,k,t}$ that encoded the decision to start a particular request at a particular time on a particular antenna. This leads to quadratic penalty functions to avoid invalid schedules, such as a request being fulfilled multiple times (2) or two conflicting activities on a single antenna (3). The state space of the custom solver is, essentially, a Gantt chart of a feasible schedule like in Fig. 3.

The driving Hamiltonian in the custom solver is actually an algorithm that encodes how one steps from one Gantt chart to a "nearby" one, akin to how one flips one bit in the state space of a QUBO. In the QUBO formalism, flipping a bit represents an attempt to assign a starting time to a currently unfulfilled request, or deallocate a currently scheduled request (hoping it later will be assigned a more effective starting time). The analogue of this for the custom solver is to deallocate the request if applicable, then select a few view periods and attempt to fit the request into the resulting Gantt chart during one of those windows. If this is not possible, then that request remains unfulfilled with the hope that as other requests are moved it will later find a more effective slot.

In practice, we use a "soft scheduling" mechanism. By far the most challenging constraint to satisfy in this approach is ensuring that no two activities on the same antenna overlap in time. When scheduling a request, rather than assigning it a specific activity period we assign it the largest possible time window consistent with the selected view period. If this time window conflicts with time windows already assigned to the antenna, we can attempt to shrink the time windows of all the affected requests to fit in the new request. As long as we do not shrink any time windows shorter than the minimum requested activity times, then we may assign the new request onto the antenna. When we deallocate a request, we can expand the time windows of the remaining requests assigned to that antenna, hence recovering this soft scheduling flexibility for a later assignment attempt.

As a consequence, the solver does not produce the final Gantt chart, but rather one where each scheduled request obtains a time window long enough to hold its activity period. The actual assignment of activity periods into the time window is done in post-processing. At this point one could enforce any secondary desiderata; for this problem there were none and so we simply centered each activity period in the assigned time window.

Substochastic Monte Carlo is a population algorithm that interleaves the kinetics, expressed through the stepping described above, and a birth/death process governed by the potential energy (objective function). The standard quantum annealing schedule is to start the process with strong kinetics and weak potential and evolve to one with no kinetics and a strong potential. Mathematically, one can renormalize time so that the kinetics remains at a constant strength, while the potential increases over the course of the algorithm, which is the approach we take.

Each member of the population, a "walker," is a Gantt chart of a feasible schedule. We initialize a walker with a greedy algorithm: randomly order the requests and view periods, and schedule as many as possible. Initially, only a modest number of requests get fulfilled.

The optimization algorithm then proceeds over a number of "epochs" as follows.

1) First, each member of the population takes several steps, attempting to allocate or move a requests based on its internal Gantt chart. This forms the kinetic, or diffusion, part of the algorithm.
2) Then the entire population undergoes Gibbs sampling: each walker is given a utility E according to how many requests its Gantt chart fulfills, and progresses to the next epoch with likelihood $\frac{1}{Z}e^{\beta E}$ where β is the strength of the potential energy for this epoch and Z is a normalization factor.

That is, a walker whose Gantt chart fulfills more requests is exponentially more likely to survive the Gibbs sampling process.

In practice, we want the population size P to be constant over the course of the algorithm and so we use a more deterministic method for selecting walkers to survive: we assign each walker a weight $\frac{P}{Z}e^{\beta E}$, and select P new walkers according to this weighting. In this way, if $\frac{P}{Z}e^{\beta E} \geq 1$ then such a walker is guaranteed to appear at least $\lfloor \frac{P}{Z}e^{\beta E} \rfloor$ times in next epoch, while if $\frac{P}{Z}e^{\beta E} < 1$ then this walker has this probability of surviving the sampling.

One can show that as this algorithm runs, the population forms an empirical sample that approximates the quasi-stationary distribution of the quantum ground state, albeit normalized in L_1 as opposed to quantum states that are L_2-normal. Consequently, this classical diffusion algorithm exhibits behavior akin to quantum tunneling: when walkers get trapped in an unfavorable part of the state space they eventually die in favor of cloning walkers in better positions, and so the walker appears to have tunneled across the state space onto the site of another walker [28].

The current implementation of this custom solver is in Python 3.8, utilizing the `multithreading` package for basic parallelization. In the first stage of each epoch, each of the P walkers independently evaluate the assignment/deallocation of a request based on its current Gantt chart. As this is "embarrassingly parallel" as commented above, we may perform this part of the algorithm on different threads. Based on some basic parameter optimization, we found the ideal number of walkers is roughly 50–100 for problems of this size, hence we could potentially match the population size to the number of threads available and hence achieve good parallelization for this component of the algorithm. The second phase of each epoch involves assembling the utilities of each walker's Gantt chart, and sampling based on this, which is not suitable for parallelization, and hence, done in serially on the main thread.

V. EXTENDED QUBO FORMULATION

A. USER SATISFACTION

As mentioned in the previous section, the quality of a schedule can also be evaluated with a quantity that measures user satisfaction. To this end, Mark Johnston has introduced two metrics in [24]. The first one is the overall root mean squared unsatisfied time fraction

$$U_{rms} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{T_{R_i} - T_{S_i}}{T_{R_i}}\right)^2} \tag{5}$$

with $T_{R_i} = \sum_{k\in A_i} t_{R_k}$, $T_{S_i} = \sum_{k\in A_i} t_{S_k}$, A_i set of requests for mission i and t_{R_k} (t_{S_k}) the requested (scheduled) time for request k. Another metric introduced in [24] is the following:

$$U_{max} = \max_{i\in\{1\ldots N\}}\left(\frac{T_{R_i} - T_{S_i}}{T_{R_i}}\right) \tag{6}$$

which indicates the worst case (max) unsatisfaction of any individual mission, where 1.0 means no requested time for that mission was included in the schedule.

Those metrics can be evaluated after a schedule is generated or, alternatively, be included as constraint in the cost function, which would enforce a given level of user satisfaction is achieved during the optimization. We implemented a version of U_{rms} that was compatible with a QUBO formulation by adopting a pseudo ℓ^1 norm instead of a ℓ^2 norm. The difference $T_{R_i} - T_{S_i}$ being always positive, there is no need to take the absolute value of the difference. Minimizing the following metric would increase user satisfaction:

$$U_{us} = \sum_{m\in\mathcal{M}} \frac{1}{T_{R_m}} \sum_{i\in A_m} t_{R_i} - t_{S_i}. \tag{7}$$

A term $(t_{R_i} - t_{R_i})x_i$ would not penalize a solution when a request would not be selected ($x_i = 0$). To retain the penalizing terms t_{R_i} for unscheduled requests, the binary variable x selecting a given request is applied to the scheduled time $t_{R_i} - t_{R_i}x_i$ and not the entire time difference. The term t_{R_i} being a constant with respect to the binary variable, minimizing the metric U_{us} (7) is achieved by minimizing

$$h_{us}(\bar{x}) = -\sum_{m\in\mathcal{M}} \frac{1}{T_{R_m}} \sum_{i\in A_m} t_{S_i}x_i. \tag{8}$$

Minimizing U_{us} is, thus, equivalent to maximizing the scheduled time per mission.

Since we do not know before hand what T_{S_m} is for each mission m, the formulation should include a term that constrains it to be the sum of all scheduled tracks during the optimization. To this end, we follow a procedure described by Andrew Lucas [29]. The main idea is to introduce a new binary variable y that selects one integer value T_{S_m} in the range 1 to a maximum value T_{R_m}, which is known before the computation begins: $T_{S_m} = \sum_{n=1}^{T_{R_m}} ny_n$. When y_n=1 for one value of n and all the other $y_{n'}$=0 for all $n' \neq n$, $T_{S_m} = n$. We must complement (8) with the following term in order to implement the metric (7) in QUBO formulation:

$$h_{us_2}(\bar{x}) = \sum_{m\in\mathcal{M}} \left(\sum_{n=1}^{T_{R_m}} ny_n - \sum_{i\in A_m} t_{S_i}x_i\right)^2. \tag{9}$$

However, the maximum scheduled time for a request is about 10 h, or 36,000 s. There can be several dozens of requests per mission. As a consequence, the formulation above could add several 100,000 variables per mission and therefore be

Algorithm 2: Binary Variables Generation for User Satisfaction Cost Function.

```
input  : A request list $\{request\}$
output: A binary variable list $\{y\}$
// Create mission list $\{mission\}$:
for request in $\{request\}$ do
    $mission \leftarrow mission[request]$
end
// Set maximum exponent $M$:
for mission in $\{mission\}$ do
    $M \leftarrow \lfloor log_2 T_R \rfloor$
end
// Create binary variables $\{y\}$:
for mission in $\{mission\}$ do
    for all bits $n$ in $T_R$ binary expansion do
        if $n < M$ then
            $y \leftarrow (mission, M, 2^n)$
        else
            $y \leftarrow (mission, M, T_R + 1 - 2^M)$
        end
    end
end
```

unpractical. Andrew Lucas [29] indicated an alternative approach that uses a binary expansion of an integer number N. This expansion uses $M + 1$ binary variables instead of N with $M = \lfloor log_2 N \rfloor$, drastically reducing the number of y variables needed. In order to satisfy the user satisfaction metric (7), we implemented the QUBO terms (8) and (9) using this binary expansion and added those two terms to the total cost function (4). We detail the transformation of DSN information into binary variables y with the pseudocode of Algorithm 2.

We performed several experiments and found quite logically that it was not possible to improve the user satisfaction without sacrificing other metrics (number of request satisfied, number of conflicts,...). Using an hyper-parameter equal to 1.43 in (9) and 1.19 in (8) and data from week 44 of 2016, the rms mean squared unsatisfied time fraction U_{rms} decreased from 31.15% (no user satisfaction constraint, in (4) alone to 29.31%. The number of requests satisfied went down to 244 from the 245 satisfied when no user satisfaction was unforced. The total scheduled time went from 975.90 h to 991.40 h. Fig. 6 shows the number of requests and time scheduled difference between the two implementations.

The results for other missions not shown in Fig. 6 were identical for both implementations. The number of variables increased relatively little to 60,105 from the value indicated in Table 2 while the number of finite QUBO elements of the Q matrix more than tripled to reach 147,960,129. This last number is commensurate with the maximum numbers of elements allowed by D-Wave HSS, two hundred millions, and thereby gives an indication of the kind of problem size and sophistication that can be currently solved with this

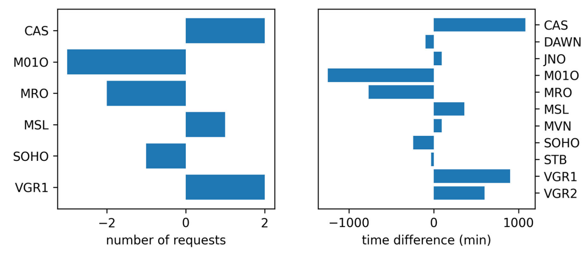

FIGURE 6. Difference between results obtained with, and without, a user satisfaction constraint implemented in the QUBO cost function, for week 44 of 2016. A positive quantity indicates that the user satisfaction constrain improved (increased) it. Conversely, a negative quantity indicates the quantity was greater when no user satisfaction was part of the cost function.

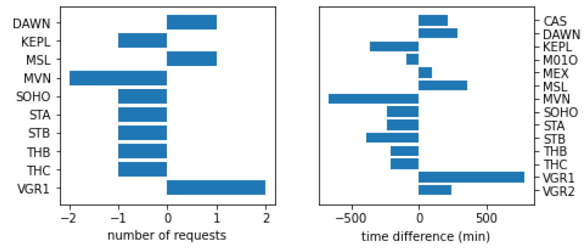

FIGURE 7. Difference between results obtained with, and without, a priority constraint implemented in the QUBO cost function, for week 44 of 2016. A positive quantity indicates that the priority constrain improved (increased) it. Conversely, a negative quantity indicates the quantity was greater when no priority constraint was part of the cost function.

service. Including the user satisfaction metric (7) as a constraint to optimize during the computation did produce a solution schedule with a lower unsatisfied time fraction than the formulation where this constraint was not present, thereby demonstrating the efficiency of our QUBO formulation of a user satisfaction constraint. This improved user satisfaction was achieved by increasing the total scheduled time.

B. PRIORITIES

In this section, we explore the flexibility of the QUBO formulation by taking advantage of the formal resemblance of a list of scheduled times, constrained by the DSN requests, and a list of request priorities, that could all be adjusted independently by a scheduler. We note that we can accommodate for user defined request priorities by replacing t_{S_i} in (8) by a priority p_i for request i

$$h_P(\bar{x}) = \sum_{m\in\mathcal{M}} \frac{1}{P_m} \sum_{i\in A_m} p_i x_i \tag{10}$$

with $P_m = \sum_{i\in A_m} p_i$. We adopt the following convention: request i has a higher priority than a request j if $p_i < p_j$. We implemented this equation into a new QUBO formulation and added a term similar to (9), h_{P_2}, by replacing T_{R_m} by P_m and t_{S_i} by a priority p_i in (9). Likewise, the pseudo code describing the transformation of DSN information into binary variables y necessary for the calculation of h_{P_2} is the same as Algorithm 2 while replacing T_{R_m} by P_m.

We use the request list from week 44, 2016 to illustrate the performance of this formulation. For this particular week, the Voyager 1 mission, VGR1, was the mission that had the least amount of requested time satisfied resulting in a U_{max} of 89.2% when scheduled solely using (4). In order to promote Voyager 1 mission, we set the priority of each of its requests to 1 while setting the priorities of all other requests to 10. The hyper-parameters were set to 1.43 and 1.23 for the h_P and h_{P_2} terms, respectively. The total number of variables was 59 802. As it was the case in the previous section, the addition of the two QUBO terms using the binary representation recommended by Andrew [29] resulted in a relatively modest increase of the number of variables (198) compared to the simple formulation of (4). The number of finite elements in the QUBO matrix Q was 147 329 864. Similarly to the previous section, the additional terms implementing the priority constraint more than tripled the number of elements compared to the number indicated in Table 2. Fig. 7 shows the difference in the number of requests and time scheduled between the formulation with and without priority constraint.

The increase in both the number of requests and number of time scheduled for the Voyager 1 mission is clear. The priority formulation works well and can enable a user to prioritize a mission independently from the time and resource constraints. Enforcing a prioritization of the Voyager 1 mission comes at the cost for other missions. There are less request scheduled (240) with the additional QUBO terms implementing the prioritization than without (245, see Table 4). The root mean squared unsatisfied time fraction U_{rms} is higher when prioritizing Voyager 1, 37.3%, than when not prioritizing this mission (31.5%).

For the past few years, the introduction of request priorities has been considered for the DSN scheduling [3], [5], [24]. The new prioritization scheme would shift the primary goal from eliminating conflicts, to maximizing the weighted degree of satisfaction of user requirements. This new prioritization scheme is still under development and not used in operations. We emphasize that prioritization method introduced in this section is distinct from the DSN prioritization discussed in [3], [5], and [24]. Our goal in this section was to use the formal equivalence between the scheduled time t_{S_i} in (8) and a priority (weight) freely adjustable to give user the ability to adjust individual request priorities independently and independently from the time and resource constraints of the requests.

VI. CONCLUSION

In this work, we described a QUBO formulation of the DSN scheduling problem. This formulation attempts to maximize the number of requests satisfied while minimizing the number of conflicts between scheduled activities. It allows for the shortening of long tracks, the scheduling of multiple-antenna (2) simultaneously and for the splitting of long tracks into two smaller tracks when the request time is longer than 8 h. The DSN scheduling process operating on a rolling week basis, we tested our formulation with two different weeks of user requests. Given our choice of problem characteristics such as time granularity, this inputs resulted in QUBO matrices with several tens of thousands of variables. These problem sizes being too large for current quantum annealers, we resorted to using D-Wave HSS.

We compared our results with those of Claudet et al. [26] who used the same data and produced DSN schedules with a conventional MILP technique. Solutions returned by the HSS with our QUBO matrices satisfied substantially more requests, scheduled more hours and had lower unsafistied time fraction than those obtained with MILP using a fifteen time longer run time than the HSS run time.

We implemented a QUBO formulation to minimize unsatisfied time fraction during the computation of the solutions. This extended formulation succeeded in scheduling more hours in week 44, 2016 than when this constraint was not used, resulting in a lower unsatisfied time fraction.

Using the formal equivalence between a request scheduled time and a request priority, we formulated a QUBO constraint that allows the user to set each request priority independently for each request. This should be contrasted with requests scheduled times that are highly constrained by the problem characteristics (time and resource constraints) and therefore interdependent. We demonstrated the use of this extended formulation by assigning higher priorities to Voyager 1 mission requests, which indeed resulted in a higher number of satisfied requests and higher scheduled time for this mission.

Both the user satisfaction and priority constraints increased the number of QUBO matrix elements to about 150 millions elements, a number commensurate with the current maximum number of elements D-Wave's HSS can handle (200 millions). This proximity sets a limit on the QUBO formulation sophistication of the deep space network scheduling problem that can be achieved in the near future. This work demonstrated promising prospects of the QUBO formulation to solve the DSN scheduling problem but more work would be needed to accommodate constraints not taken into account in this work (and listed here [2]).

We also described a custom solver approach using quantum-inspired optimization. This solver could not handle a specific type of request, and hence scheduled fewer requests overall than the HSS approach. Yet, among those requests it could treat it produced relatively comparable schedules, in terms of antenna utilization, 9–10 times faster than the HSS solution (even after excluding the time to produce the QUBO). While the custom solver can easily be modified to treat more complex objectives and request priorities, these can be a challenge for QUBO formulations.

In recent years, there has been a revived interest in quadratic unconstrained binary optimization due to the commercial availability of quantum annealers using this formulation and the prospects that this quantum technology could eventually provide an advantage over conventional computing. In a technical report on D-Wave HSS and Advantage chip, McGeoch et al. [19] demonstrated a *quantum acceleration* using the HSS compared to situations where only the QPU or classical solver were used. Two elements were necessary for this demonstration. First, access to same classical solver used in the HSS is needed for comparison. Second, a large amount of available time on the QPU since the demonstration involved several trial of about 15-h-long runs. We did not have either so we could not assess the role of the quantum hardware in the performance of our computation. We note that [19] is a technical report published by D-Wave. We are not aware of any peer-reviewed publications that could duplicate the results described in [19] and discuss those results in terms of D-Wave HSS 2 design characteristics. In this context, a possible future direction for this work could include the development of a classical software that could solve QUBO problems and that we could utilize in tandem with a QPU within a customized hybrid framework (using `dwave-hybrid` for instance). This would provide us with the opportunity to assess the solver's performance in terms of the hybrid framework design and evaluate the relative contribution of the classical and quantum part of the computation.

A fast, efficient, and fair optimization of the initially oversubscribed schedule could save some days of time in the mid-range process, and correspondingly reduce the cost. The optimization described in this work represents an important step toward achieving this goal.

VI. ACKNOWLEDGMENT

The authors would like to thank J. Lazio, S. A. Townes, B. Castano, and C. Mattmann for providing the support for this work. The authors would like to thank the Quantum Artificial Intelligence Laboratory (QuAIL) team, D. Venturelli, E. Rieffel, and D. Bell for letting them access D-Wave 2000Q system at NASA Ames during the first year of the project (fall 2018 to late 2019). This access was instrumental to the progress of the work before D-Wave commercial Leap service was available. The authors would also like to thank S. Pakin for providing access to the Los Alamos National Laboratory D-Wave 2000Q system. This access proved very helpful when the authors did not have access to another quantum annealer early 2020, when the world went into a lockdown due to the COVID-19 pandemic and before they could procure an access to D-Wave Leap service. The authors would also like to thank S. Severini and E. Kessler for providing access to AWS Braket preview and to M. Schuetz and G. Salton of AWS Quantum Solutions Laboratory for introducing them to Meta-Analytics, a startup company offering a classical QUBO solver, Alpha-QUBO. The authors would also like to thank all the Meta-Analytics team, F. Glover, G. Kochenberger, R. Hennig, and R. Chawla for providing access to Alpha-QUBO which allowed to benchmark their formulation with an early version of the DSN input. Part of this research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

REFERENCES

[1] M. Johnston et al., “Automating mid- and long-range scheduling for NASA’s deep space network,” in *Proc. Int. Conf. Space Oper.*, 2012, pp. 327–351, doi: 10.2514/5.9781624102080.0327.0352.

[2] M. D. Johnston et al., “Automated scheduling for NASA’s deep space network,” *AI Mag.*, vol. 35, no. 4, pp. 7–25, Dec. 2014, doi: 10.1609/aimag.v35i4.2552.

[3] C. Shouraboura, M. D. Johnston, and D. Tran, “Prioritization and oversubscribed scheduling for NASA’s deep space network,” in *Proc. Scheduling Plan. Appl. Workshop Int. Conf. Automated Plan. Scheduling*, 2016.

[4] M. D. Johnston and J. Lad, “Integrated planning and scheduling for NASA’s deep space network—from forecasting to real-time,” in *Proc. Int. Conf. Space Oper.*, 2018, doi: 10.2514/6.2018-2728.

[5] M. D. Johnston, “User preference optimization for oversubscribed scheduling of NASA’s deep space network,” in *Proc. 11th Int. Workshop Plan. Scheduling Space*, 2019, pp. 86–92.

[6] S. Chien et al., “ASPEN - Automating space mission operations using automated planning and scheduling,” in *Proc. Int. Conf. Space Oper.*, 2000.

[7] C. E. Bell, “Scheduling deep-space network data transmissions: A Lagrangian relaxation approach,” *Proc. SPIE*, vol. 1963, Mar. 1993, pp. 330–340, doi: 10.1117/12.141750.

[8] M. D. Johnston, “Multi-objective scheduling for NASA’s future deep space network array,” in *Proc. Int. Workshop Plan. Scheduling Space*, 2006.

[9] M. Brown and M. D. Johnston, “Experiments with a parallel multi-objective evolutionary algorithm for scheduling,” in *Proc. 8th Int. Workshop Plan. Scheduling Space*, 2013.

[10] A. Guillaume et al., “Deep space network scheduling using evolutionary computational methods,” in *Proc. IEEE Aerosp. Conf.*, 2007, pp. 1–6, doi: 10.1109/AERO.2007.352900.

[11] E. Goh, H. Venkataram, M. Hoffmann, M. D. Johnston, and B. D. Wilson, “Scheduling the NASA deep space network with deep reinforcement learning,” in *Proc. IEEE Aerosp. Conf.*, 2021, pp. 1–10, doi: 10.1109/AERO50100.2021.9438519.

[12] M. W. Johnson et al., “Quantum annealing with manufactured spins,” *Nature*, vol. 473, no. 7346, pp. 194–198, May 2011, doi: 10.1038/nature10012.

[13] C. McGeoch and P. Farré, “The D-wave advantage system: An overview,” D-Wave, Burnaby, BC, Canada, Tech. Rep., 2020. [Online]. Available: https://www.dwavesys.com/media/s3qbjp3s/14-1049a-a_the_d-wave_advantage_system_an_overview.pdf

[14] T. Inagaki et al., “A coherent Ising machine for 2000-node optimization problems,” *Science*, vol. 354, no. 6312, pp. 603–606, Nov. 2016, doi: 10.1126/science.aah4243.

[15] K. Kim et al., “Quantum simulation of frustrated ising spins with trapped ions,” *Nature*, vol. 465, no. 7298, pp. 590–593, Jun. 2010. doi: 10.1038/nature09071.

[16] I. Mahboob, H. Okamoto, and H. Yamaguchi, “An electromechanical Ising Hamiltonian,” *Sci. Adv.*, vol. 2, no. 6, Jun. 2016, doi: 10.1126/sciadv.1600236.

[17] H. Goto, K. Tatsumura, and A. R. Dixon, “Combinatorial optimization by simulating adiabatic bifurcations in nonlinear Hamiltonian systems,” *Sci. Adv.*, vol. 5, no. 4, Apr. 2019, doi: 10.1126/sciadv.aav2372.

[18] *Toshiba, Simulated Bifurcation Machine (SBM) User Manual* Revision 1.20,Tokyo, Japan: Toshiba Digital Solutions Corp., 2019.

[19] C. McGeoch, P. Farré, and W. Bernoudy, “D-wave hybrid solver service advantage: Technology update,” D-Wave Inc., Burnaby, BC, Canada, Tech. Rep., 2020. [Online]. Available: https://www.dwavesys.com/media/m2xbmlhs/14-1048a-a_d-wave_hybrid_solver_service_plus_advantage_technology_update.pdf

[20] V. N. Smelyanskiy et al., “A near-term quantum computing approach for hard computational problems in space exploration,” 2012, *arXiv:1204.2821*.

[21] D. Venturelli, D. J. J. Marchand, and G. Rojo, “Job shop scheduling solver based on quantum annealing,” 2016, *arXiv:1506.08479v2*.

[22] M. D. Johnston, D. Tran, B. Arroyo, and C. Page, “Request-driven scheduling for NASA’s deep space network,” in *Proc. Int. Workshop Plan. Scheduling Space*, 2009. [Online]. Available: https://doi.org/10.2514/6.2010-2265

[23] M. D. Johnston and D. Tran, “The DSN scheduling engine (DSE): Automated scheduling of activities for the NASA deep space network,” in *Proc. Int. Symp. Artif. Intell. Robot. Automat. Space*, 2010.

[24] M. D. Johnston, “Scheduling NASA’s deep space network: Priorities, preferences, and optimization,” in *Proc. Int. Conf. Automated Plan. Scheduling Scheduling Plan. Appl. Workshop*, 2020.

[25] E. Goh, H. Venkataram, B. Balaji, M. D. Johnston, and B. Wilson, “SatNet: A benchmark for satellite scheduling optimization,” in *Proc. AAAI-22 Workshop Mach. Learn. Oper. Res.*, 2021. [Online]. Available: https://openreview.net/forum?id=buIUxK7F-Bx

[26] T. Claudet, R. Alimo, E. Goh, M. D. Johnston, R. Madani, and B. D. Wilson, “Δ-MILP: Deep space network scheduling via mixed-integer linear programming,” *IEEE Access*, vol. 10, pp. 41330–41340, 2021, doi: 10.1109/ACCESS.2022.3164213.

[27] M. Jarret, S. P. Jordan, and B. Lackey, “Adiabatic optimization versus diffusion Monte Carlo methods,” *Phys. Rev. A*, vol. 94, no. 4, Art. no. 0 42318, 2016, doi: 10.1103/PhysRevA.94.042318.

[28] M. Jarret and B. Lackey, “Substochastic Monte Carlo algorithms,” 2017, *arXiv:1704.09014*.

[29] L. Andrew, “Ising formulations of many NP problems,” *Front. Phys.*, vol. 2, 2014, Art. no. 5, doi: 10.3389/fphy.2014.00005.

Alexandre Guillaume received the B.S., M.S., and Ph.D. degrees from Joseph Fourier University, Grenoble, France, in 1995 and 1999, respectively, all in physics.

From 2001 to 2002, he was a Caltech Postdoctoral Research Associate with the Jet Propulsion Laboratory (JPL), California Institute of Technology, Pasadena, CA, USA. From 2003 to 2004, he was a Postdoctoral Research Associate working on theoretical schemes to achieve the Heisenberg limit with superconducting quantum sensors with JPL, where he has been a Data Scientist with the Science Data Modeling and Computing group since 2007. He has worked on statistical analysis, physical modeling, and machine learning techniques applied to remote sensing, planetary science, operational research, with a main focus on Earth Science.

Edwin Y. Goh, photograph and biography not available at the time of publication.

Mark D. Johnston, photograph and biography not available at the time of publication.

Brian D. Wilson, photograph and biography not available at the time of publication.

Anita Ramanan, photograph and biography not available at the time of publication.

Frances Tibble, photograph and biography not available at the time of publication.

Brad Lackey, photograph and biography not available at the time of publication.