Received February 15, 2021, accepted March 6, 2021, date of publication March 9, 2021, date of current version March 17, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3064928

Deep Space Network Scheduling via Mixed-Integer Linear Programming

ALEX SABOL[1], RYAN ALIMO[2], FARHAD KAMANGAR[1], AND RAMTIN MADANI[3]
[1]Computer Science and Engineering Department, The University of Texas at Arlington, Arlington, TX 76015, USA
[2]Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109, USA
[3]Electrical Engineering Department, The University of Texas at Arlington, Arlington, TX 76015, USA

Corresponding author: Ramtin Madani (ramtin.madani@uta.edu)

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

ABSTRACT NASA's Deep Space Network (DSN) is a globally-spanning communications network responsible for supporting the interplanetary spacecraft missions of NASA and other international users. The DSN is a highly utilized asset, and the large demand for its' services makes the assignment of DSN resources a daunting computational problem. In this paper we study the DSN scheduling problem, which is the problem of assigning the DSN's limited resources to its users within a given time horizon. The DSN scheduling problem is oversubscribed, meaning that only a subset of the activities can be scheduled, and network operators must decide which activities to exclude from the schedule. We first formulate this challenging scheduling task as a Mixed-Integer Linear Programming (MILP) optimization problem. Next, we develop a sequential algorithm which solves the resulting MILP formulation to produce valid schedules for large-scale instances of the DSN scheduling problem. We use real world DSN data from week 44 of 2016 in order to evaluate our algorithm's performance. We find that given a fixed run time, our algorithm outperforms a simple implementation of our MILP model, generating a feasible schedule in which 17% more activities are scheduled by the algorithm than by the simple implementation. We design a non-MILP based heuristic to further validate our results. We find that our algorithm also outperforms this heuristic, scheduling 8% more activities and 20% more tracking time than the best results achieved by the non-MILP implementation.

INDEX TERMS Optimization, optimization methods, scheduling.

I. INTRODUCTION

NASA's Deep Space Network (DSN) is a globally-spanning communications network responsible for supporting the interplanetary spacecraft missions of NASA and other international users [1]. NASA's DSN is one of the largest and most sensitive telecommunications systems in the world, consisting of three large antenna facilities spaced approximately 120 degrees apart, allowing for constant communication between the DSN's ground stations and the spacecraft they service [1], [2]. Located in Goldstone, CA, Madrid, Spain, and Canberra, Australia, each DSN complex contains one massive 70-meter antenna and up to four 34-meter antennas [1]–[3]. The DSN supports a multitude of users across a wide variety of scientific missions, including lunar, earth orbiting, ground-based, and deep space missions [3]. As of April 2019, the DSN supports 70 unique missions [4], offering a broad range of services including telemetry, spacecraft command, tracking, radio astronomy, and more [1]. The large variety of services provided by the DSN means that requirements from mission to mission can vary drastically, making the assignment of DSN resources a difficult logistical problem.

The DSN is a highly utilized asset, and NASA has estimated that deep space communication capabilities will need to be expanded by a factor of 10 each decade in order to keep up with the rising demands [5]. NASA spends a considerable amount of resources on its DSN, for example in 2014 the DSN commanded a budget of \$210 million [6]. Despite impressive throughput and growing demand, the DSN budget has dropped by approximately 10 million dollars per year since 2014 [6]. With the increasing demand and a decreasing budget, efficient use of the DSN assets is of utmost importance as we move forward into the future.

In broad terms, DSN scheduling is the process of allocating DSN resources to the missions that request them.

The associate editor coordinating the review of this manuscript and approving it for publication was Shih-Wei Lin.

The numerous operational constraints and large scale makes DSN scheduling a challenging logistical task. This paper considers the process of DSN scheduling over a one week time horizon. Currently, this process begins with missions submitting communication requests to the DSN. The DSN then uses automated tools to generate an initial suboptimal schedule, which generally contains conflicts. This schedule is then manually modified by a DSN expert who removes as many of these conflicts as possible. Next, the schedule is released back to the missions, who engage in a peer-to-peer negotiation process to eliminate the remaining conflicts. Once the negotiations end, the schedule is finalized and published. The large demand placed on the network makes resource allocation nontrivial, and has led to the investigation of automatic methods to help DSN operators construct optimal schedules.

The motivation for this work is twofold: First, the DSN scheduling process is bottlenecked by the need for extensive manual conflict removal by DSN operators and users after the initial schedule is generated. If we can generate a higher quality schedule initially, much of the overhead spent manually refining the schedule could be eliminated. Second, the DSN scheduling process is strikingly similar to several other problems in the satellite scheduling domain, but comparison is difficult as DSN scheduling has not been fully described using a formal mathematical framework. We develop a mathematical model of the DSN scheduling process to elucidate its relationship to other scheduling problems, and also to illustrate the nuances of DSN scheduling which justify the construction of our model.

A. LITERATURE REVIEW

The automation of DSN scheduling has an extensive history. In 1992 a Lagrangian Relaxation approach, LR-26, was developed by Bell for supporting automated scheduling of 26-meter antennas [7]. A generic spacecraft scheduling framework, ASPEN, was created by NASA in 1997 [8], and its integration into the DSN scheduling process began in 2000 [9]. ASPEN utilizes a well-known technique called iterative repair - an infeasible schedule is first generated, then conflicts are identified and eliminated one by one, until the schedule contains no conflicts. In 2006, a genetic algorithm for DSN scheduling was proposed [10], and in 2008 a generalized differential evolutionary algorithm was investigated to deal with the multi-objective nature of the DSN scheduling problem [11]. By 2009, DSN scheduling had moved to a request-driven paradigm, and the DSN scheduling Engine (DSE) was developed, which is a distributed architecture utilizing multiple instances of ASPEN simultaneously [12]. More recently, Hackett, Bilen, and Johnston proposed a novel 'block scheduling' algorithm, which leverages spacecraft positioning in order to reduce the amount of setup and teardown time scheduled [13]. The majority of the aforementioned works focus on operational details of DSN scheduling, and do not present a unified mathematical model that can be used as the basis for future studies. The linear programming models of DSN scheduling [7], [14] that are be found in the literature are quite dated, and do not take into consideration important features of the DSN scheduling process such as setup/teardown times and activity splitting. Access to increased computational power coupled with the development of sophisticated commercial solvers over the decades since these papers were written has made optimization-based schemes more tractable, and reignited interest in MILP formulations of the DSN scheduling process. We address these issues by developing a concise model of the DSN scheduling process, with the goal of connecting DSN scheduling with the broader scheduling literature. In this work we significantly extend ideas presented in our previously published conference paper [15]. The model presented in [15] does not allow activities to be split into smaller segments, and does not scale well to real world problem sizes. In this work we introduce a new MILP formulation which incorporates these features, and also scales much better than the previous model.

A mixed integer linear program (MILP) is an optimization problem in which a linear objective is optimized over a set of affine constraints, with some of the variables restricted to be integers. Mixed integer linear programming is well-known to be NP-hard in general [16], making it well suited for modeling difficult optimization problems involving discrete decision making such as the DSN scheduling problem. Scheduling problems are particularly well-suited for mixed integer linear programming formulations, and numerous examples of MILP based scheduling can be found in almost every field of science and engineering. For instance, real-time scheduling of precedence-constrained task graphs is discussed in [17], [18]. In [19] the authors use a MILP model to maximize revenue by the automated scheduling of an energy hub in a thermal energy market. In [20], [21] the authors discuss the flexible job shop and hybrid flow shop scheduling problems respectively, in the context of optimizing workshop energy consumption. Pipeline pump scheduling optimization using MILP formulations is studied in [22], [23].

An important difference between DSN scheduling and other scheduling problems found in the literature is its oversubscription. The DSN receives many more requests than it can accommodate, which makes the completion time based objective functions commonly found in the scheduling literature lack meaning. Oversubscribed, overloaded, or overconstrained scheduling problems arise naturally in many domains such as airport gate assignment [24], processor task scheduling [25], [26], and meeting scheduling [27], [28]. Perhaps most similar to the DSN scheduling problem are problems in the satellite scheduling domain, including satellite range scheduling [29], [30], telescope scheduling [31], [32], payload scheduling [33], [34], and in particular the satellite observation scheduling problem [35]–[41], which also includes the time window constraints encountered in DSN scheduling. Although the DSN scheduling problem is similar to many of the oversubscribed scheduling problems present in the literature, it has several unique properties. In particular, the ability to 'split' activities into smaller operational segments, as well as the interplay between time windows and required setup/teardown time for each activity motivated our decision to create an entirely new model rather than expanding upon an existing model from the satellite scheduling literature.

DSN scheduling is a complicated iterative process, making it difficult to describe in a formal manner. In order for complex real-world processes such as DSN scheduling to be studied from a theoretical standpoint, it is critical to develop mathematical models which are amenable to quantitative research. One of the goals of this paper is to develop a mathematical model which incorporates many of the important constraints encountered in the real world DSN scheduling process. By studying this model, we can gain insight into the benefit of pursuing automation.

Another goal of this work is to construct an algorithm which supports a central DSN operator by quickly producing an initial conflict free schedule. The main contributions of this paper can be summarized as follows:

- Formulating the Deep Space Network scheduling problem as a mixed integer linear program, the solution of which is a valid schedule that satisfies the many operational constraints present in the DSN scheduling problem.
- In order to improve the scalability of this MILP based approach, we develop a sequential algorithm which introduces randomization in order to aid the solver in exploring more of the search space, as well as break some of the symmetries in the problem which lead to many redundant branches by the branch and bound solver.
- We evaluate our algorithm's performance over several experiments on real-world data from the DSN scheduling problem. In addition, we develop a greedy heuristic to serve as a baseline for comparison to further validate our algorithm's efficacy.

The remainder of the paper is organized as follows. In section II the main characteristics of the DSN scheduling problem are summarized, and the MILP formulation is given. In section III we introduce an algorithm to solve the proposed MILP. In section IV we present experimental results. Finally, concluding remarks are discussed in section V.

B. NOTATIONS

$\mathbb{R}$, $\mathbb{N}$, $\mathbb{Z}_+$ denote the sets of real numbers, natural numbers, and nonnegative integers respectively. We use Calligraphic uppercase letters to denote sets, boldface uppercase letters to denote matrices, and boldface lowercase to denote vectors. $|\mathcal{X}|$ denotes the cardinality of a set $\mathcal{X}$. Given a vector $\boldsymbol{v}$, $\text{diag}\{\boldsymbol{v}\}$ denotes a diagonal square matrix whose diagonal entries are the elements of $\boldsymbol{v}$. The superscript $(\cdot)^T$ denotes the transpose operator. $\mathbf{0}_{|\mathcal{X}|}$, $\mathbf{1}_{|\mathcal{X}|}$ denote a $|\mathcal{X}|$ dimensional vector of zeros, ones respectively. $\mathbf{0}_{|\mathcal{X}|\times|\mathcal{Y}|}$, $\mathbf{1}_{|\mathcal{X}|\times|\mathcal{Y}|}$ denote an $|\mathcal{X}| \times |\mathcal{Y}|$ matrix of zeros, ones respectively. $\{0,1\}^{|\mathcal{X}|\times|\mathcal{Y}|}$, $\{0,1\}^{|\mathcal{X}|}$ denote the set of $|\mathcal{X}| \times |\mathcal{Y}|$ matrices, and $|\mathcal{X}|$ dimensional vectors respectively, whose entries are zero or one.

II. PROBLEM DESCRIPTION

A. PROBLEM STATEMENT

In the DSN scheduling process we must schedule a given set of *activities* to a given set of *resources*, over a given time horizon. Each of these activities is associated with a *mission*, and each mission services a particular active spacecraft whose location is continuously changing. Activities correspond to communication requests between these spacecraft and the DSN resources. As these resources have a fixed location at one of the DSN's ground stations they do not have constant visibility of each spacecraft, and communication can only occur within specific time windows, which we refer to as *viewperiods*. There are often many viewperiods associated with each activity, and several different resources with valid viewperiods for each activity.

Each activity has a duration, which is divided into three components: setup time, tracking time, and teardown time. This is important because only the tracking time is required to be scheduled within a valid viewperiod, the setup and teardown time are not constrained to occur within a valid viewperiod. The setup and teardown times are non-negotiable, but tracking time may be adjusted if needed. Each DSN resource is constrained by its bandwidth, and can only communicate with a limited number of spacecraft at any given time.

In order to fit more activities into the schedule, the DSN allows activities to be split into smaller segments which do not have to be scheduled contiguously. For example, an activity with 6 hours of tracking time may be split into two 3 hour segments and scheduled separately. Activities cannot be split in an arbitrary manner, for each activity a minimum segment duration as well as a minimum downtime required between each segment. As a concrete example, an activity with 6 hours of tracking time may require that each segment be at least 2 hours long, with 1 hour of downtime required between each segment. The caveat to splitting activities in this fashion is that each segment requires setup and teardown time, making split activities more costly to schedule.

The decentralized nature of the DSN scheduling process makes it difficult to represent the objective as a single function. The missions operate independently of one another, and each mission would like the DSN to schedule as many of their own activities as possible. As a result, a valid schedule which is acceptable from the perspective of one mission may leave the other missions dissatisfied. On the other hand, the DSN wishes to use their resources efficiently, and has concerns such as maximizing the overall resource usage and minimizing resource downtime. Thus, a straightforward objective which satisfies both the DSN and its users is maximizing the number of activities scheduled. This is easily extendable to a weighted objective, such as maximizing the amount of tracking time scheduled.

1) ASSUMPTIONS

In order to incorporate the operational details described above into our problem formulation, we build a model of the DSN scheduling process based on the following assumptions:

1) Each resource can only support one activity at any given time.
2) Each activity can only be scheduled to one viewperiod at any given time.
3) If an activity is scheduled, its tracking time must occur within a valid viewperiod.
4) If an activity is scheduled, it must be scheduled for at least its minimum requested tracking time, and at most its maximum requested tracking time.
5) An activity requires a certain amount of setup time before tracking time can occur, and teardown time after.
6) Activities may be 'split' into segments, i.e. not scheduled contiguously.
    a) If an activity is split, setup and teardown is required for each segment.
    b) If an activity is split, there is a minimum duration of a segment.
    c) If an activity is split, there is a minimum time separation between segments.

B. MATHEMATICAL FORMULATION

We seek to assign a given set of activities $\mathcal{A}$ to a set of resources $\mathcal{R}$ as efficiently as possible, throughout a discrete planning horizon which is divided into a set of time epochs $\mathcal{T} = \{1, 2, \dots, |\mathcal{T}|\}$. To this end we need to define the set of viewperiods $\mathcal{V} \subseteq \mathcal{A} \times \mathcal{R}$. A pair (a, r) belongs to $\mathcal{V}$ if and only if there exists $t \in \mathcal{T}$ during which the activity $a \in \mathcal{A}$ can be assigned to the resource $r \in \mathcal{R}$. To streamline the presentation, we first define a number of binary matrices as follows:

- *Viewperiod-resource incidence matrix* $\boldsymbol{R} \in \{0, 1\}^{|\mathcal{R}|\times|\mathcal{V}|}$: This matrix maps viewperiods to their corresponding resources, i.e., $R_{r,v} = 1$ iff $v \in \mathcal{V}$ is associated with $r \in \mathcal{R}$.
- *Viewperiod-activity incidence matrix* $\boldsymbol{A} \in \{0, 1\}^{|\mathcal{A}|\times|\mathcal{V}|}$: This matrix maps viewperiods to their corresponding activities, i.e., $A_{a,v} = 1$ iff $v \in \mathcal{V}$ is associated with $a \in \mathcal{A}$.
- *Time-viewperiod incidence matrix* $\boldsymbol{V} \in \{0, 1\}^{|\mathcal{V}|\times|\mathcal{T}|}$: For every $v = (a, r) \in \mathcal{V}$ and $t \in \mathcal{T}$, we have $V_{v,t} = 1$ iff the activity a can be assigned to the resource r during the epoch t.

We define the following vectors that impose certain practical limits on resources:

- *Minimum and maximum tracking duration vectors* $\boldsymbol{d}_{\min}, \boldsymbol{d}_{\max} \in \mathbb{Z}_+^{|\mathcal{A}|}$: These vectors, respectively, encapsulate the minimum and maximum time that should be spent on a scheduled activity. In other words, we say an activity $a \in \mathcal{A}$ is *scheduled* if it has been assigned at least its minimum tracking duration, $\boldsymbol{d}_{\min,a}$, and at most its maximum tracking duration, $\boldsymbol{d}_{\max,a}$, while no time should be spent on an unscheduled activity.
- *Setup and teardown duration vectors* $\boldsymbol{\delta}_{\uparrow}, \boldsymbol{\delta}_{\downarrow} \in \mathbb{Z}_+^{|\mathcal{V}|}$: These vectors encapsulate the amount of time necessary to prepare each resource to engage in and disengage from each activity, respectively.
- *Minimum up and down time vectors* $\boldsymbol{\gamma}_{\uparrow}, \boldsymbol{\gamma}_{\downarrow} \in \mathbb{N}^{|\mathcal{V}|}$: These vectors contain the minimum uninterrupted time that should be spent on each activity, once it is started and the amount of time that the resource should remain idle after the activity is interrupted.
- *Minimum up and down time vectors* $\boldsymbol{\gamma}_{\uparrow}, \boldsymbol{\gamma}_{\downarrow} \in \mathbb{N}^{|\mathcal{V}|}$: These vectors contain the minimum uninterrupted time that should be spent on each activity, once it is started and the amount of time that the resource should remain idle after the activity is interrupted.
- *Viewperiod start and end time vectors* $\boldsymbol{s}, \boldsymbol{e} \in \mathcal{T}^{|\mathcal{V}|}$: These vectors contain the time epoch in which each viewperiod starts and ends, respectively. A viewperiod $v \in \mathcal{V}$ starts at time s_v, and ends at time e_v.

Lastly, the main variables of the optimization problem are as follows:

- $\boldsymbol{X} \in \{0, 1\}^{|\mathcal{V}|\times|\mathcal{T}|}$: For every viewperiod $v = (a, r) \in \mathcal{V}$ and epoch $t \in \mathcal{T}$, we have $X_{v,t} = 1$, iff the resource r is allocated to the activity a during epoch t.
- $\boldsymbol{X}^{\uparrow} \in \{0, 1\}^{|\mathcal{V}|\times|\mathcal{T}|}$: For every viewperiod $v = (a, r) \in \mathcal{V}$ and epoch $t \in \mathcal{T}$, we have $X^{\uparrow}_{v,t} = 1$, iff the resource r starts working on activity a during epoch t, i.e.,
$$X^{\uparrow}_{v,t} = 1 \iff X_{v,t-1} = 0 \ \wedge\ X_{v,t} = 1 \tag{1}$$
- $\boldsymbol{X}^{\downarrow} \in \{0, 1\}^{|\mathcal{V}|\times|\mathcal{T}|}$: For every viewperiod $v = (a, r) \in \mathcal{V}$ and epoch $t \in \mathcal{T}$, we have $X^{\downarrow}_{v,t} = 1$, iff the resource r stops working on the activity a during epoch t, i.e.,
$$X^{\downarrow}_{v,t} = 1 \iff X_{v,t-1} = 1 \ \wedge\ X_{v,t} = 0 \tag{2}$$
- $\boldsymbol{Y}^{\uparrow} \in \{0, 1\}^{|\mathcal{V}|\times|\mathcal{T}|}$: For every viewperiod $v = (a, r) \in \mathcal{V}$ and epoch $t \in \mathcal{T}$, we have $Y^{\uparrow}_{v,t} = 1$, iff the resource r is in the process of setting up to start the activity a during epoch t, i.e.,
$$Y^{\uparrow}_{v,t} = 1 \iff \exists\, t < \tau \le t + \delta^{\uparrow}_{v} : \quad X^{\uparrow}_{v,\tau} = 1 \tag{3}$$
- $\boldsymbol{Y}^{\downarrow} \in \{0, 1\}^{|\mathcal{V}|\times|\mathcal{T}|}$: For every viewperiod $v = (a, r) \in \mathcal{V}$ and epoch $t \in \mathcal{T}$, we have $Y^{\downarrow}_{v,t} = 1$, iff the resource r is in the process of tearing down after disengaging from the activity a during epoch t, i.e.,
$$Y^{\downarrow}_{v,t} = 1 \iff \exists\, t - \delta^{\downarrow}_{v} < \tau \le t : \quad X^{\downarrow}_{v,\tau} = 1 \tag{4}$$
- $\boldsymbol{x} \in \{0, 1\}^{|\mathcal{A}|}$: For every $a \in \mathcal{A}$, we have $x_a = 1$ iff the activity a is completed throughout the planning horizon.

The two objective functions we are most interested in, *number of activities scheduled* and *amount of tracking time scheduled*, respectively, are defined below as:
$$f_0(\boldsymbol{x}) := \boldsymbol{1}_{|\mathcal{A}|}^{\top}\boldsymbol{x} \tag{5}$$
$$f_1(\boldsymbol{X}) := \sum_{v\in\mathcal{V}}\sum_{t\in\mathcal{T}} X_{v,t} \tag{6}$$

For the sake of presentation we contain our optimization variables in a tuple $\boldsymbol{Z}$, i.e.

$$\boldsymbol{Z} := (\boldsymbol{X},\ \boldsymbol{X}^{\uparrow},\ \boldsymbol{X}^{\downarrow},\ \boldsymbol{Y}^{\uparrow}, \boldsymbol{Y}^{\downarrow},\ \boldsymbol{x})$$

Using the aforementioned notations, the DSN scheduling problem can be cast in the following form:

$$\begin{aligned}
&\underset{\boldsymbol{Z}}{\text{maximize}}\ \ f_0(\boldsymbol{x}) && && \text{(7a)}\\
&\text{subject to} && &&\\
&\boldsymbol{X} \le \boldsymbol{V} && && \text{(7b)}\\
&X_{v,t} - X_{v,t-1} = X^{\uparrow}_{v,t} - X^{\downarrow}_{v,t} && \forall v \in \mathcal{V},\ \ \forall t \in \mathcal{T} && \text{(7c)}\\
&\sum_{\tau=t-\gamma_v^{\uparrow}+1}^{t} X^{\uparrow}_{v,\tau} \le X_{v,t} && \forall v \in \mathcal{V},\ \ \forall t \in \mathcal{T} && \text{(7d)}\\
&\sum_{\tau=t-\gamma_v^{\downarrow}+1}^{t} X^{\downarrow}_{v,\tau} \le 1 - X_{v,t} && \forall v \in \mathcal{V},\ \ \forall t \in \mathcal{T} && \text{(7e)}\\
&Y^{\uparrow}_{v,t} = \sum_{\tau=t+1}^{t+\delta_v^{\uparrow}} X^{\uparrow}_{v,\tau} && \forall v \in \mathcal{V},\ \ \forall t \in \mathcal{T} && \text{(7f)}\\
&Y^{\downarrow}_{v,t} = \sum_{\tau=t+1-\delta_v^{\downarrow}}^{t} X^{\downarrow}_{v,\tau} && \forall v \in \mathcal{V},\ \ \forall t \in \mathcal{T} && \text{(7g)}\\
&\boldsymbol{R}(\boldsymbol{Y}^{\downarrow} + \boldsymbol{X} + \boldsymbol{Y}^{\uparrow}) \le \boldsymbol{1}_{|\mathcal{R}|\times|\mathcal{T}|} && && \text{(7h)}\\
&\text{diag}\{\boldsymbol{d}_{\min}\}\boldsymbol{x} \le \boldsymbol{A}\boldsymbol{X}\boldsymbol{1}_{|\mathcal{T}|} \le \text{diag}\{\boldsymbol{d}_{\max}\}\boldsymbol{x} && && \text{(7i)}
\end{aligned}$$

where

- (7a) accounts for the total number of activities that are scheduled.
- (7b) enforces the possibility of assigning activities to resources based on viewperiods.
- (7c) enforces (1) and (2).
- (7d) and (7e) impose the minimum up and down time constraints.
- (7f) and (7g) impose the definition of $\boldsymbol{Y}^{\downarrow}$ and $\boldsymbol{Y}^{\uparrow}$, i.e., (3) and (4).
- (7h) ensures that each resource is only assigned to one activity at a time. Moreover, this constraint ensures sufficient room for setup and teardown procedures.
- (7i) imposes the minimum and maximum duration for scheduled activities.

III. PROPOSED ALGORITHM

We solved (7a) – (7i) using commercial solvers; in our simulations we observed two problematic behaviors which our algorithm is designed to alleviate. In a fixed run time simulation on the order of several hours, the solver made most of its progress in improving the solution quality within the first few minutes of the simulation. Second, the solver 'plateaus', experiencing long periods of time with no improvement to the objective value.

Algorithm 1 is an iterative algorithm which solves a modified instance of (7a) – (7i) in each iteration and stores the resulting solution for use in subsequent iterations. In each iteration a randomized objective function is generated, and the optimal solution with respect to this new objective function is chosen out of the pool previously obtained solutions and used as an initial point to solve (7a) – (7i) with the new objective function. We limit the time spent solving the MILP each iteration using a local time limit. The iterative phase is terminated by a global time limit, after which (7a) – (7i) is solved a final time using the original objective $f_0(\boldsymbol{x})$ and the best possible initial point from the pool of solutions produced during the iterative phase. We now describe the details of our objective function randomization method.

A. RANDOMIZATION OF OBJECTIVE FUNCTION

In our original formulation, each activity scheduled is rewarded equally by the objective function. A traditional branch and bound algorithm struggles with highly symmetrical problems, so by introducing random weights to the objective function we break some of the problem's symmetry and avoid unnecessary branching. Additionally, randomized weights allow the solver to potentially escape local optima which it may have otherwise been trapped by. To this end, we employ a randomized objective function

$$f_{\boldsymbol{c}_1,\boldsymbol{c}_2}(\boldsymbol{x}, \boldsymbol{X}) = \boldsymbol{c}_1^{\top}\boldsymbol{x} + \boldsymbol{c}_2^{\top}\boldsymbol{X}\boldsymbol{1}_{|\mathcal{T}|} \tag{8}$$

where the elements of the coefficient vectors $\boldsymbol{c}_1 \in \mathbb{R}^{|\mathcal{A}|}$ and $\boldsymbol{c}_2 \in \mathbb{R}^{|\mathcal{V}|}$ are uniformly chosen from the intervals $[1, 5]$ and $[0, 0.01]$, respectively. The coefficient $\boldsymbol{c}_1$ imposes an artificial priority on the activities, eliminating many cases where the solver must choose between two activities which are rewarded identically by the original objective function. The coefficient $\boldsymbol{c}_2$ allows the solver to also reward tracking time, in contrast with $f_0(\boldsymbol{x})$, which only rewards the number of activities scheduled.

We use several new notations when describing Algorithm 1. For the sake of brevity, we slightly abuse notation by leaving out set inclusion when assigning variables. For example, the assignment '$\boldsymbol{Z} \leftarrow \boldsymbol{0}$' is understood to mean

$$\boldsymbol{Z} \leftarrow (\boldsymbol{0}_{|\mathcal{V}|\times|\mathcal{T}|}, \boldsymbol{0}_{|\mathcal{V}|\times|\mathcal{T}|}, \boldsymbol{0}_{|\mathcal{V}|\times|\mathcal{T}|}, \boldsymbol{0}_{|\mathcal{V}|\times|\mathcal{T}|}, \boldsymbol{0}_{|\mathcal{V}|\times|\mathcal{T}|}, \boldsymbol{0}_{|\mathcal{A}|}).$$

global_TIMELIMIT and local_TIMELIMIT are user imposed parameters which limit Algorithm 1's total run time, and the time spent each iteration solving the MILP, respectively. Additionally, we define $\boldsymbol{Z}_k$, $\check{\boldsymbol{Z}}$, and $\boldsymbol{Z}_{\text{f}}$ as the solution ($\boldsymbol{Z}$) obtained at iteration k, the temporary initial point used, and the solution obtained by using the best initial point found during the iterative phase, respectively.

IV. NUMERICAL RESULTS

We have performed several experiments to demonstrate the efficacy of our model. We limit our study to the real world DSN data set for week 44 of 2016, which consists of 14 resources, 286 activities, 1430 hours of requested tracking time, and 30 different missions. In Table 1 we provide a summary of the parameters associated with the 286 activities in this data set. Each row corresponds to a mission, and contains the number of activities, n, associated with this mission,

Algorithm 1: Randomized MILP

```
- $Z_0 \leftarrow 0$
- $k \leftarrow 1$
while $time < global\_TL$ do
    - Select the the elements of $\boldsymbol{c}_1 \in \mathbb{R}^{|\mathcal{A}|}$ and $\boldsymbol{c}_2 \in \mathbb{R}^{|\mathcal{V}|}$
      uniformly from the intervals [1, 5] and [0, 0.01], respectively.
    -$\check{\boldsymbol{Z}} \leftarrow \underset{i=0,1,\ldots,k-1}{\arg\max} \; f_{\boldsymbol{c}_1,\boldsymbol{c}_2}(\boldsymbol{x}_i, \boldsymbol{X}_i)$
    - Solve problem (7a) – (7i) with the revised objective
      function $f_{\boldsymbol{c}_1,\boldsymbol{c}_2}$ and initial point $\check{\boldsymbol{Z}}$ to obtain $\boldsymbol{Z}_k$,
      the best feasible solution obtained within local_TL.
    - $k \leftarrow k + 1$
end
-$\check{\boldsymbol{Z}} \leftarrow \underset{i=0,1,\ldots,k-1}{\arg\max} \; f_0(\boldsymbol{x}_i)$
- Solve problem (7a) – (7i) with the original objective
  function $f_0(\boldsymbol{x})$ and initial point $\check{\boldsymbol{Z}}$ and return $\boldsymbol{Z}_f$, the best
  feasible solution obtained within local_TL.
```

as well as parameter values for these activities. We provide the average minimum and maximum duration, specified in hours, as well as the average setup and teardown time, specified in minutes. All MILP formulations are solved using GUROBI 9.0.2 or CPLEX 12.10.0 through MATLAB R2020a on a laptop computer with Intel Xeon six-core 2.8 GHz CPU, and 32 GB of RAM.

The first experiment is designed with the goal of deciding which solver will be used to solve our MILP formulations in the following experiments. We chose two state of the art commercial solvers, GUROBI and CPLEX, for this comparison. In our simulation, we solve (7a) – (7i) using each solver with default parameter settings and a fixed run time of 54000 seconds. We found that GUROBI significantly outperformed CPLEX, achieving higher values for both of our objective functions of interest, f_0(number of activities scheduled) and f_1(number of tracking time scheduled). To be more specific, GUROBI scheduled 212 out of the 286 activities, and 78.6% of the requested time. On the other hand, CPLEX scheduled 120 activities and 60.3% of the requested tracking time. The schedules produced by both CPLEX and GUROBI are shown in Figures 1 and 2, respectively.

In the next experiment, we examine how the choice of time discretization affects the solver. Recall that we divide our planning horizon of one week into discrete time epochs. In our formulation (7a) – (7i) each viewperiod v has several variables associated with each time epoch, which means that the number of variables in the MILP problem is dependent on the size of the time epoch. On one hand, a smaller time discretization gives the solver more flexibility while scheduling, meaning that better schedules may be possible at smaller time discretizations. On the other hand, decreasing the time discretization increases the number of variables, which causes the solver to take more time to generate a high

TABLE 1. Average parameter values organized by mission for the week 44, 2016 DSN data set.

Mission	n	$d_{\min}$(hrs)	$d_{\max}$(hrs)	$\delta_{\uparrow}$(mins)	$\delta_{\downarrow}$(mins)
ACE	10	2.75	2.75	60	15
CAS	8	7.2	9	60	15
CHDR	21	1	1	60	15
DAWN	21	6.4	8	60	15
DSCO	1	1	1	60	15
GTL	21	1.1	1.1	30	15
HYB2	2	3	3	60	15
JNO	18	5.95	7.11	63.33	16.67
KEPL	2	6	6	60	15
LRO	13	1	1	60	15
M01O	14	6.4	8	60	15
MER1	7	2.5	2.5	45	15
MEX	7	6.4	8	60	15
MMS1	13	3.94	3.94	60	15
MOM	8	3.19	3.19	46.88	15
MRO	14	6.4	8	60	15
MSL	7	6	6	60	15
MVN	9	6.4	8	60	15
NHPC	7	8	10	60	15
ORX	10	4.5	4.5	60	15
SOHO	21	4	4	60	15
STA	7	4	4	60	15
STB	3	5	5.17	45	15
STF	5	4	4	60	15
THB	4	3.5	3.5	60	15
THC	4	3.5	3.5	60	15
VGR1	8	6.67	8.13	46.88	15
VGR2	11	6.41	7.68	35.45	15
WIND	7	2.5	2.5	60	15
XMM	3	5	5	60	15

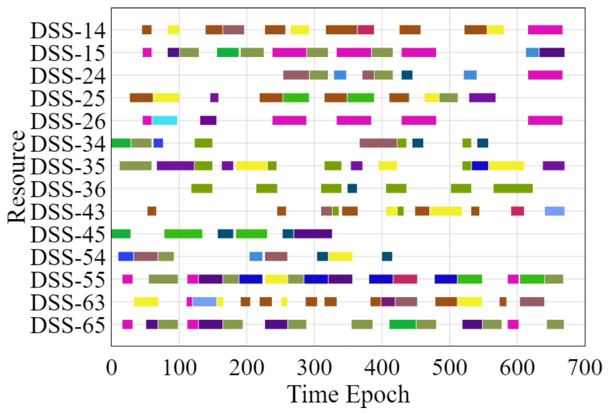


FIGURE 1. Schedule produced using MILP model for week 44 of 2016 with CPLEX solver and default parameter settings. 120 out of 286 activities scheduled. Colors are representative of distinct sets of activities, regarded as missions.

quality solution. The choice of time discretization is also influenced by the dataset. As shown in Table 1, the average teardown time for the majority of the activities in the data set is 15 minutes, and in fact the minimum teardown time is 15 minutes. This limits our choice in time discretization. For example, if an activity requires 15 minutes of teardown time, and we divide our planning horizon into 30 minute time epochs, we will be wasting 15 minutes when scheduling this

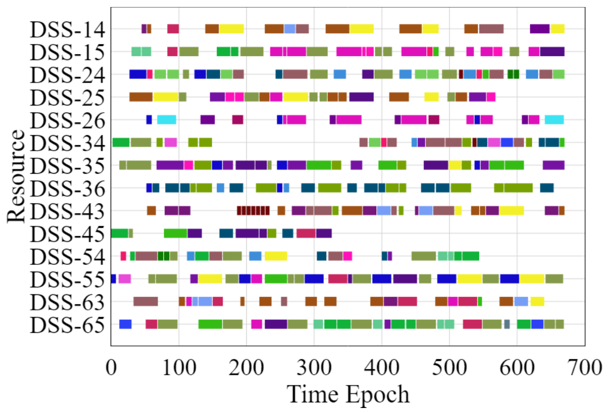


FIGURE 2. Schedule produced using MILP model for week 44 of 2016 with GUROBI solver and default parameter settings. 212 out of 286 activities scheduled. Colors are representative of distinct sets of activities, regarded as missions.

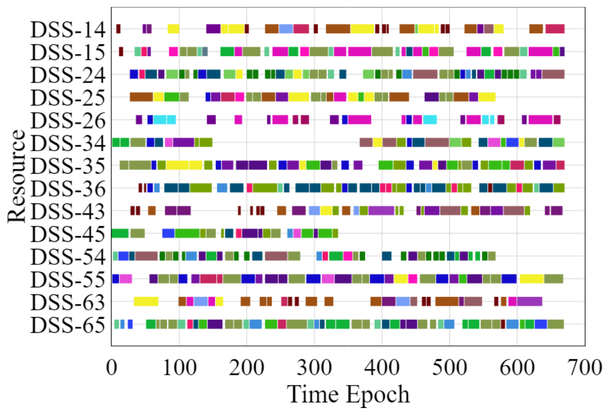


FIGURE 3. Schedule produced using Algorithm 1, with random objective weights for week 44 of 2016. 250 out of 286 activities scheduled. Colors are representative of distinct sets of activities, regarded as missions.

TABLE 2. Objective values obtained by the simple MILP implementation using different time discretizations.

Discretization (mins)	Tracking Time(hrs)	Activities Scheduled
5	1069.25	180
15	1124.25	212
30	1160	253

teardown time, as the smallest possible assignment of one time epoch corresponds to 30 minutes of real time. Thus, we would like the time discretization to be small enough that time is not wasted on setup/teardown, but large enough that the solver is able to find a high quality solution in a reasonable amount of time. We test three choices of time discretization:

- 5 minute intervals ($|\mathcal{T}| = 2016$),
- 15 minute intervals ($|\mathcal{T}| = 672$), and
- 30 minute intervals ($|\mathcal{T}| = 336$).

The simulation consisted of solving (7a) – (7i) for each of the 3 time discretizations using GUROBI with a fixed run time of 54000 seconds. The results have been summarized in Table 2. As expected, the number of activities scheduled by GUROBI increased as the time discretization increased, demonstrating the solver's difficulty in handling the larger problem size. The solution produced in the 30 minute case is surprisingly good considering the earlier discussion of overhead generated when scheduling the teardown time. We chose to use a time discretization of 15 minutes for the remaining experiments, as a 30 minute discretization creates a large amount of unavoidable overhead, and with a 5 minute discretization, the solver struggles to produce a high quality solution in a reasonable amount of time.

In the next experiment, we compare the results obtained by simply solving the MILP formulation (7a)-(7i) using GUROBI with a time limit of 54000 seconds to the results obtained using Algorithm 1 with the same time limit. We use a time limit of 300 seconds to control the amount of time spent solving the MILP on each iteration of Algorithm 1. By using Algorithm 1 we are able to schedule significantly more activities than the simple MILP implementation.

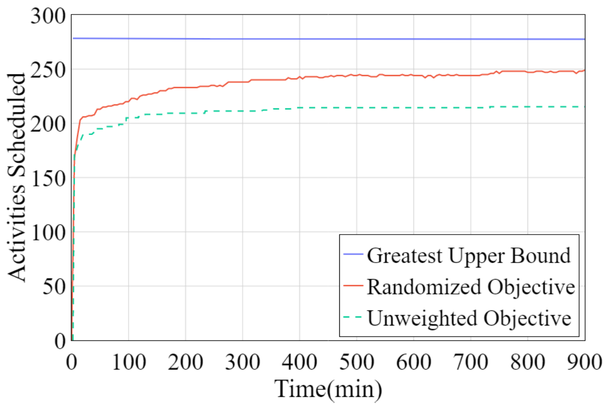


FIGURE 4. Solver progress vs Time. Notice that the randomized algorithm has a higher objective value at all times, and continues to make progress while the unweighted MILP plateaus.

Algorithm 1 schedules 250 out of the 286 activities, while the simple MILP implementation schedules 212 activities within the same amount of time. The objective value f_0 is not monotonically increasing while using Algorithm 1, as can be seen in Figure 4. Nevertheless, Algorithm 1 achieves and maintains a greater value for the objective f_0 than the simple MILP implementation throughout the entire simulation. As the objective f_1 is not explicitly incorporated into our formulation, the simple MILP implementation remains competitive with respect to the amount of tracking time scheduled. Nevertheless, Algorithm 1 schedules 1126 hours of tracking time, slightly outperforming the simple MILP implementation which scheduled 1124.25 hours of tracking time. The schedule generated using Algorithm 1 can be seen in Figure 3, whereas the schedule generated by the simple MILP implementation is shown in Figure 2.

A. ALTERNATIVE METRICS

The decentralized nature of DSN scheduling makes user satisfaction an important consideration when evaluating the efficacy of our algorithm. For each user, $u \in \mathcal{U}$, we define

$$\text{User Satisfaction} = \frac{t_u^{sched}}{t_u^{req}} \tag{9}$$

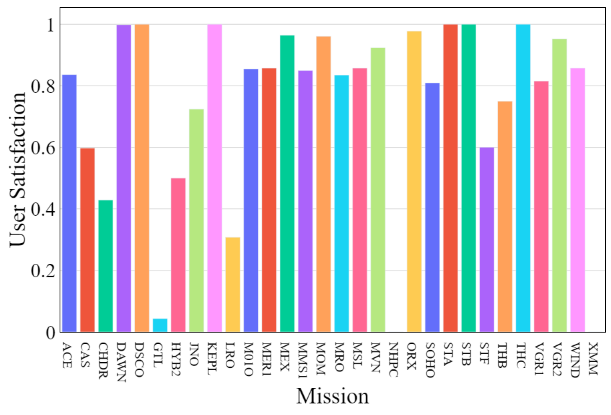


FIGURE 5. User Satisfaction achieved using the simple MILP implementation.

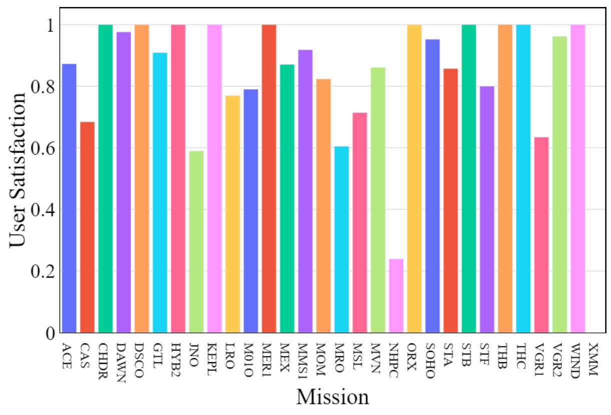


FIGURE 6. User Satisfaction achieved using Algorithm 1.

where t_u^{sched} and t_u^{req} are the number of time epochs assigned to user u and the number of time epochs requested by user u, respectively. User satisfaction for a simple implementation of our MILP as well as by using Algorithm 1 can be seen in Figures 5 and 6, respectively. Another metric which can be used to judge the strength of our output schedules is the root mean square unsatisfied time fraction, which we define as

$$U_{RMS} = \sqrt{\frac{1}{|\mathcal{U}|}\sum_{u\in\mathcal{U}}\left(\frac{t_u^{req} - t_u^{sched}}{t_u^{req}}\right)^2} \tag{10}$$

Ideally, every user is scheduled their requested time and $U_{RMS} = 0$. On the other hand, the worst case scenario occurs when no requested time is scheduled, in which case $U_{RMS} = 1$. In reality, U_{RMS} is somewhere between zero and one, as we can schedule some but not all of the activities. Using Algorithm 1 we obtain a U_{RMS} of.28, whereas a value of.4 is obtained by the simple MILP implementation, showing that an average DSN user is more satisfied by the schedule obtained using Algorithm 1 than the schedule produced by a simple MILP implementation.

B. BASELINE HEURISTICS

In the previous experiments we show that Algorithm 1 outperformed the simple MILP implementation with respect to the main objectives f_0 and f_1, and furthermore, Algorithm 1 produced higher quality solutions with respect to the alternative metrics introduced in the previous section. However, both schemes incorporated (7a) – (7i), meaning additional evidence is needed to justify the MILP model. To further evaluate Algorithm 1 as well as our MILP model, we designed a heuristic scheme for DSN scheduling that does not involve the MILP formulation. Our heuristic combines two common strategies in greedy scheduling heuristics: scheduling in order of duration and scheduling in order of start time. We test three schemes: scheduling shortest duration activity first, scheduling longest duration activity first, and scheduling the activities in a random order. In order to incorporate activity splitting into the heuristic, we construct the schedule in two distinct phases. As splitting activities into segments costs additional setup and teardown time, it makes sense to limit the amount of splitting done when generating the schedule. To accomplish this we first assign all activities that do not require splitting. In other words, we first fill the schedule with as many full duration activities as possible. Next, we are left with a set of unscheduled activities which cannot fit into a single viewperiod and must be split into segments. To avoid unnecessary setup and teardown time, we schedule the segments to the longest duration viewperiods first. This is in contrast to phase 1, where we selected the earliest viewperiod first. The heuristic scheme used in this paper is detailed as follows.

Step 1: Sort the activities according to choice of heuristic. We investigate three different sorting paradigms in this paper: Sorting in ascending order of activity length(shortest activity first), Sorting in descending order of activity length(longest activity first), and sorting randomly(random activity first).

Step 2: Select the first activity i from the sorted list $\mathbf{L}$. Select the earliest viewperiod j associated with this activity. Search the interval $[s_j - \delta_j^{\uparrow}, e_j - \delta_j^{\downarrow}]$ for an available time window of length $\delta_j^{\uparrow} + \boldsymbol{d}_{\min,i} + \delta_j^{\downarrow}$. If such a time window is found, add activity i to the schedule, remove activity i from $\mathbf{L}$, and mark this time window unavailable. Otherwise, select the next earliest viewperiod associated with activity i, and repeat this process. Once all viewperiods associated with activity i have been checked, repeat this step for the next activity in $\mathbf{L}$.

Step 3: The remaining activities in $\mathbf{L}$ were unable to be scheduled into one contiguous time window, which means they must be 'split' into smaller segments and scheduled using multiple viewperiods. Select the first activity i from $\mathbf{L}$. Select the longest duration viewperiod j associated with activity i. Within this viewperiod, mark all available time windows with length greater than $\delta_j^{\uparrow} + \delta_j^{\downarrow}$ as tentatively unavailable. Repeat this process for each viewperiod. Let ns$_i$ be the number of time windows we have tentatively scheduled for activity i. If $\boldsymbol{d}_{\min,i} + ns_i(\delta_j^{\uparrow} + \delta_j^{\downarrow})$ has been tentatively scheduled, add activity i to the schedule, remove i from $\mathbf{L}$, and mark the tentatively scheduled time windows as unavailable, otherwise mark these time windows as available. Repeat this step for the next activity in $\mathbf{L}$.

TABLE 3. Objective values obtained using heuristic schemes. For each objective, the most optimal value has been highlighted in bold.

Heuristic		Shortest	Longest	Random
f_0	min.	223	205	212
	max.	**232**	216	230
	av.	227.8	210.4	222.5
	stdev.	1.6	1.9	2.7
f_1	min.	842.25	874	852
	max.	905.75	927.5	**936**
	av.	876.65	898.3	893.15
	stdev.	38.3	31.6	50.5
U_{RMS}	min.	.3644	.3886	**.3321**
	max.	.4089	.4637	.4272
	av.	.3863	.4254	.3785
	stdev.	.0066	.0114	.0150

We have designed this heuristic scheme such that it produces viable schedules which do not violate any of the constraints present in the DSN scheduling problem, thus we will be left with a feasible but not necessarily optimal schedule upon completion. To summarize, we sort our activities in step 1. In step 2 we schedule as many activities contiguously as possible, and in step 3 we fill in the gaps in the schedule by allowing the remaining unscheduled activities to be split into smaller segments. Note that in step 2 we search the viewperiods in order of start time, but in step 3 we search the viewperiods in order of duration. We do this because a split activity requires setup and teardown time for each segment, so by scheduling longer segments we reduce the number of segments needed, thus reducing the amount of setup and teardown time required.

In the final experiment we evaluate the schedules generated using the above heuristic scheme for each of the three activity sorting paradigms described above. In contrast with our MILP implementations, this heuristic procedure takes less than a second, making a run time based stopping criterion unnecessary. We perform 1000 simulations for each activity sorting paradigm. The results of this experiment are summarized in Table 3. For each of the three heuristics we report the minimum, maximum, average, and standard deviation obtained for each objective function over 1000 trials. The objective values for f_1, the amount of tracking time scheduled, are reported in hours. We name the heuristics based on the activity sorting method used in Step 1, and use the labels 'Shortest', 'Longest', and 'Random' to refer to the shortest activity first, longest activity first, and random activity first sorting methods, respectively. For all three activity sorting paradigms, the heuristic generates competitive results with respect to f_0, but performs poorly with respect to f_1. Out of the three non-MILP heuristics, the shortest activity first heuristic performs the best with respect to f_0, scheduling 232 activities. This is higher than the f_0 value of 212 achieved by the simple MILP implementation, but lower than the value achieved by Algorithm 1, which schedules 250 activities. The standard deviation of f_0 is relatively low for all three non-MILP heuristics, making it unlikely that a significantly improved value of f_0 can be achieved using these heuristics. The difference in schedule quality between the MILP and non-MILP heuristics is much more evident when interpreting the amount of tracking time scheduled, f_1. Out of the non-MILP heuristics, the random activity first sorting scheme achieves the highest value of f_1, scheduling 936 hours of tracking time. Both the simple MILP implementation and Algorithm 1 achieve significantly better results, scheduling 1124.25 hours and 1126 hours of tracking time, respectively. Interestingly, the best U_{RMS} of.3321 obtained by the random activity first heuristic is lower than the value of.4 obtained by the simple MILP implementation, indicating that the random activity first heuristic is capable of generating schedules with the tracking distributed more fairly between the missions. Nevertheless, Algorithm 1 produces a lower U_{RMS} of.28, outperforming all three non-MILP heuristics.

V. CONCLUSION

In this work we designed a new MILP formulation of the DSN scheduling problem. We then describe an algorithm which utilizes randomization in order to outperform a simple implementation of our MILP formulation. The proposed algorithm is applied to the real-world problem for week 44 of 2016. We show experimentally that, from the DSN's perspective, our algorithm outperforms a simple MILP implementation both in terms of the number of activities scheduled as well as the amount of tracking time scheduled. Moreover, the algorithm also performs well from the perspective of the DSN users', achieving a lower root mean square unsatisfied time fraction than the rudimentary implementation. To further validate our algorithm we designed a non-MILP based heuristic which our algorithm outperforms with respect to the number of activities scheduled, the amount of tracking time scheduled, and the root mean square unsatisfied time fraction.

ACKNOWLEDGMENT

The authors would like to thank Brian Wilson, Mark Johnston, Mark Hoffmann, Edwin Goh, Ravi Lanka, Alexandre Guillaume, Hamsa Venkataram, Sami Sahnoune, Chris Mattmann, David Hanks, Stephen Townes, Joseph Lazio, Becky Castano, Gregory Davis, and Fred Hadaegh.

REFERENCES

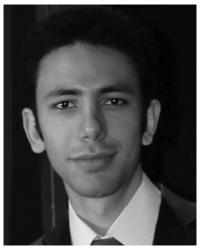
[1] J. Choi, R. Verma, and S. Malhotra, "Achieving fast operational intelligence in NASA's deep space network through complex event processing," in *Proc. 14th Int. Conf. Space Oper.*, May 2016, p. 2375.
[2] W. A. Imbriale, *Large Antennas of the Deep Space Network*, vol. 1. Hoboken, NJ, USA: Wiley, 2005.
[3] M. D. Johnston, D. Tran, B. Arroyo, S. Sorensen, P. Tay, B. Carruth, A. Coffman, and M. Wallace, "Automating mid- and long-range scheduling for NASA's deep space network," in *Proc. SpaceOps Conf.*, 2012, Art. no. 1296235.
[4] S. M. Lichten, D. S. Abraham, B. Arroyo, S. W. Asmar, J. Bell, and C. D. Edwards, "Allocation of deep space network ground system tracking and communications assets during the 2020-2021 timeframe of the 'Mars Armada,'" in *Proc. 15th Int. Conf. Space Oper.*, May 2018, p. 2502.
[5] J. Taylor, *Deep Space Communications*, vol. 1. Hoboken, NJ, USA: Wiley, 2016,
[6] "Nasa's management of the deep space network," Nat. Aeronaut. Space Admin., Office Inspector Gen., Washington, DC, USA, Tech. Rep. IG-15-013, 2015.

[7] C. E. Bell, "Scheduling deep-space network data transmissions: A Lagrangian relaxation approach," in *Applications of Artificial Intelligence 1993: Knowledge-Based Systems in Aerospace and Industry*, vol. 1963. Bellingham, WA, USA: SPIE, 1993, pp. 330–340.
[8] A. Fukunaga, G. Rabideau, S. Chien, and D. Yan, "Aspen: A framework for automated planning and scheduling of spacecraft control and operations," in *Proc. Int. Symp. AI, Robot. Automat. Space*, 1997, pp. 181–187.
[9] S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, and D. Tran, "Aspen–automated planning and scheduling for space mission operations," in *Proc. Space Ops*, 2000, pp. 1–10.
[10] A. Guillaume, S. Lee, Y.-F. Wang, H. Zheng, R. Hovden, S. Chau, Y.-W. Tung, and R. J. Terrile, "Deep space network scheduling using evolutionary computational methods," in *Proc. IEEE Aerosp. Conf.*, Mar. 2007, pp. 1–6.
[11] M. D. Johnston, "An evolutionary algorithm approach to multi-objective scheduling of space network," *Commun. Int. J. Intell. Automat. Soft Comput.*, vol. 14, no. 3, pp. 367–376, 2008.
[12] M. D. Johnston, D. Tran, and B. Arroyo, "Request-driven scheduling for NASA's deep space network," in *Proc. Int. Workshop Planning Scheduling for Space (IWPSS)*, 2009, pp. 1–10.
[13] T. M. Hackett, M. Johnston, and S. G. Bilen, "Spacecraft block scheduling for NASA's deep space network," in *Proc. Space Ops*, 2018, p. 2578.
[14] W. Webb, "An optimal spacecraft scheduling model for the NASA deep space network," *J. Brit. Interplanetary Soc.*, vol. 38, p. 422, Sep. 1985.
[15] J. A. Sabol, R. Alimo, M. Hoffmann, E. Goh, B. Wilson, and M. Johnston, "Towards automated scheduling of NASA's deep space network: A mixed integer linear programming approach," in *Proc. AIAA Scitech Forum*, Jan. 2021, p. 667.
[16] M. R. Garey and D. S. Johnson, *Computers and Intractability*, vol. 174. San Francisco, CA, USA: Freeman, 1979.
[17] S. K. Roy, R. Devaraj, and A. Sarkar, "Optimal scheduling of PTGs with multiple service levels on heterogeneous distributed systems," in *Proc. Amer. Control Conf. (ACC)*, Jul. 2019, pp. 157–162.
[18] S. K. Roy, R. Devaraj, A. Sarkar, K. Maji, and S. Sinha, "Contention-aware optimal scheduling of real-time precedence-constrained task graphs on heterogeneous distributed systems," *J. Syst. Archit.*, vol. 105, May 2020, Art. no. 101706.
[19] M. Jadidbonab, B. Mohammadi-Ivatloo, M. Marzband, and P. Siano, "Short-term self-scheduling of virtual energy hub plant within thermal energy market," *IEEE Trans. Ind. Electron.*, vol. 68, no. 4, pp. 3124–3136, Apr. 2021.
[20] L. Meng, C. Zhang, X. Shao, and Y. Ren, "MILP models for energy-aware flexible job shop scheduling problem," *J. Cleaner Prod.*, vol. 210, pp. 710–723, Feb. 2019.
[21] L. Meng, C. Zhang, X. Shao, B. Zhang, Y. Ren, and W. Lin, "More MILP models for hybrid flow shop scheduling problem and its extended problems," *Int. J. Prod. Res.*, vol. 58, no. 13, pp. 3905–3930, Jul. 2020.
[22] X. Zhou, H. Zhang, R. Qiu, Y. Liang, G. Wu, C. Xiang, and X. Yan, "A hybrid time MILP model for the pump scheduling of multi-product pipelines based on the rigorous description of the pipeline hydraulic loss changes," *Comput. Chem. Eng.*, vol. 121, pp. 174–199, Feb. 2019.
[23] Q. Liao, P. M. Castro, Y. Liang, and H. Zhang, "Computationally efficient MILP model for scheduling a branched multiproduct pipeline system," *Ind. Eng. Chem. Res.*, vol. 58, no. 13, pp. 5236–5251, Apr. 2019.
[24] H. Ding, A. Lim, B. Rodrigues, and Y. Zhu, "New heuristics for over-constrained flight to gate assignments," *J. Oper. Res. Soc.*, vol. 55, no. 7, pp. 760–768, Jul. 2004.
[25] S. K. Baruah and J. R. Haritsa, "Scheduling for overload in real-time systems," *IEEE Trans. Comput.*, vol. 46, no. 9, pp. 1034–1039, 1997.
[26] R. Al-Omari, A. K. Somani, and G. Manimaran, "Efficient overloading techniques for primary-backup scheduling in real-time systems," *J. Parallel Distrib. Comput.*, vol. 64, no. 5, pp. 629–648, May 2004.
[27] T. Tsuruta and T. Shintani, "Scheduling meetings using distributed valued constraint satisfaction algorithm," in *Proc. ECAI*, 2000, pp. 383–387.
[28] A. Lim, B. Rodrigues, R. Thangarajoo, and F. Xiao, "A hybrid framework for over-constrained generalized," *Artif. Intell. Rev.*, vol. 22, no. 3, pp. 211–243, Nov. 2004.
[29] A. J. Vazquez and R. S. Erwin, "On the tractability of satellite range scheduling," *Optim. Lett.*, vol. 9, no. 2, pp. 311–327, Feb. 2015.
[30] L. Barbulescu, J.-P. Watson, L. D. Whitley, and A. E. Howe, "Scheduling Space–Ground communications for the air force satellite control network," *J. Scheduling*, vol. 7, no. 1, pp. 7–34, Jan. 2004.
[31] M. D. Johnston, "Automated telescope scheduling," Space Telescope Sci. Inst., Baltimore, MD, USA, Tech. Rep. 306, 1988.
[32] A. I. Gómez de Castro and J. Yáñez, "Optimization of telescope scheduling: Algorithmic research and scientific policy," *Astron. Astrophys.*, vol. 403, no. 1, pp. 357–367, May 2003.
[33] S. Chien, G. Rabideau, J. Willis, and T. Mann, "Automating planning and scheduling of shuttle payload operations," *Artif. Intell.*, vol. 114, nos. 1–2, pp. 239–255, Oct. 1999.
[34] G. Rabideau, S. Chien, J. Willis, and T. Mann, "Using iterative repair to automate planning and scheduling of shuttle payload operations," in *Proc. AAAI/IAAI*, 1999, pp. 813–820.
[35] X. Chen, G. Reinelt, G. Dai, and A. Spitz, "A mixed integer linear programming model for multi-satellite scheduling," *Eur. J. Oper. Res.*, vol. 275, no. 2, pp. 694–707, Jun. 2019.
[36] G. Wu, X. Du, M. Fan, J. Wang, J. Shi, and X. Wang, "Ensemble of heuristic and exact algorithm based on the divide and conquer framework for multi-satellite observation scheduling," 2020, *arXiv:2007.03644*. [Online]. Available: http://arxiv.org/abs/2007.03644
[37] D.-H. Cho and H.-L. Choi, "A traveling salesman problem-based approach to observation scheduling for satellite constellation," *Int. J. Aeronaut. Space Sci.*, vol. 20, no. 2, pp. 553–560, Jun. 2019.
[38] Y. Xiao, S. Zhang, P. Yang, M. You, and J. Huang, "A two-stage flow-shop scheme for the multi-satellite observation and data-downlink scheduling problem considering weather uncertainties," *Rel. Eng. Syst. Saf.*, vol. 188, pp. 263–275, Aug. 2019.
[39] Y. Du, T. Wang, B. Xin, L. Wang, Y. Chen, and L. Xing, "A data-driven parallel scheduling approach for multiple agile Earth observation satellites," *IEEE Trans. Evol. Comput.*, vol. 24, no. 4, pp. 679–693, Aug. 2020.
[40] J. Li, C. Li, and F. Wang, "Automatic scheduling for Earth observation satellite with temporal specifications," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 56, no. 4, pp. 3162–3169, Aug. 2020.
[41] Y. She, S. Li, and Y. Zhao, "Onboard mission planning for agile satellite using modified mixed-integer linear programming," *Aerosp. Sci. Technol.*, vol. 72, pp. 204–216, Jan. 2018.

ALEX SABOL received the bachelor's degree in mathematics and chemistry from The University of Texas at Austin, in 2016. He is currently pursuing the Ph.D. degree in computer science with the Department of Computer Science and Engineering, The University of Texas at Arlington. His research interests include combinatorial optimization, machine learning, and data science.

RYAN ALIMO is currently a Data Scientist in machine learning with the Jet Propulsion Laboratory, California Institute of Technology. He is advancing the field of human–AI and multi-agent systems using integration of the optimization algorithms with deep learning and deploying them to spacecraft formation flying, deep-space communication, and data accountability for the Mars data.

FARHAD KAMANGAR received the Ph.D. degree in electrical engineering from The University of Texas at Arlington, in 1980. He is currently a Professor with the Department of Computer Science and Engineering, The University of Texas at Arlington.



RAMTIN MADANI received the Ph.D. degree in electrical engineering from Columbia University, New York, NY, USA, in 2015. In 2016, he was a Postdoctoral Scholar with the Department of Industrial Engineering and Operations Research, University of California, Berkeley. He is currently an Assistant Professor with the Department of Electrical Engineering, The University of Texas at Arlington, Arlington, TX, USA.