# Multirobot Onsite Shared Analytics Information and Computing

Joshua Vander Hook, Federico Rossi, *Member, IEEE*, Tiago Vaquero, Martina Troesch, Marc Sanchez Net, Joshua Schoolcraft, Jean-Pierre de la Croix, and Steve Chien

***Abstract*—Computation load sharing across a network of heterogeneous robots is a promising approach to increase robots' capabilities and efficiency as a team in extreme environments. However, in such environments, communication links may be intermittent and connections to the cloud or Internet may be nonexistent. In this article, we introduce a communication-aware, computation task-scheduling problem for multirobot systems and propose an integer linear program (ILP) that optimizes the allocation of computational tasks across a network of heterogeneous robots, accounting for the networked robots' computational capabilities and for available (and possibly time-varying) communication links. We consider scheduling of a set of interdependent required and optional tasks modeled by a dependency graph. We present a consensus-backed scheduling architecture for shared-world, distributed systems. We validate the ILP formulation and the distributed implementation in different computation platforms and in simulated scenarios with a bias toward lunar or planetary exploration scenarios. Our results show that the proposed implementation can optimize schedules to allow a three-fold increase in the amount of rewarding tasks performed (e.g., science measurements) compared to an analogous system with no computational load sharing.**

***Index Terms*—Autonomous robots, concurrency control-scheduling algorithms, distributed computing, multi robot systems, space exploration.**

## I. Introduction

MULTIAGENT systems hold great promise for science exploration in extreme environments. Correspondingly, there has been a proliferation of national programs aimed at expanding multiagent networked systems for caves [1], oceans [2], and low-earth orbit [3], [4]. These environments can be considered the "extreme edge," far from the robust computation and omnipresent communication networks of connected cities.

In planetary exploration, there is an emerging push toward more extreme environments, and therefore multiagent systems because single, flagship robots are limited to less-hostile operating areas. Therefore, access to Recurring Slope Lineae or planetary caves [5], [6] may be possible with multiple small, potentially expendable rovers. Not surprisingly, we see potential systems being demonstrated in the Mars helicopter [7], and the "PUFFER" rover (Pop-Up Flat-Folding Explorer Robots) [8]. There is also evidence that next-generation spaceflight computing employed on these systems will be more like our current mobile devices [7], [9], [10], [11], [12]. Finally, it is likely that, in the future, multiple collaborating robots, astronauts, and base stations will *themselves* be a complex and time-varying processing and communication network (see, e.g., [13]).

The key metric of these system concepts is throughput of observations and data. A compelling paradigm to increase the throughput of heterogeneous multirobot systems is *computational load sharing:* By allowing robotic agents to offload computational tasks to each other or to a "computational server" (e.g., an overhead orbiter, a flagship rover, or a stationary lander), computational load sharing can give access to advanced analysis capabilities to small, low-power rovers with limited on-board computing capabilities, or allow agents to do more memory- or CPU-intensive work by leveraging nearby idle nodes. Previous work [14], [15] has shown that computational sharing in robotic systems with heterogeneous computing capabilities (e.g., Mars exploration scenarios) can lead to significant increases in system-level performance and science returns.

These self-reliant, edge robotic systems share commonalities that motivate our study. The first is an emphasis on energy conservation due to their remote, self-sustaining design. The second is the possible use of heterogeneous systems, in which some nodes contain more resources (power, computing, communication, mobility, sensing, etc.) than others. The final factor is intermittent and periodic loss of connectivity between nodes. While the benefit of edge computation supporting mobile phone networks continues to be well investigated (see the highly influential [16]), the intermittent loss of communications and time-varying position of the agents make it more challenging to employ these concepts directly. This is true because it is challenging to route through a changing network, but also because the source and destination change over time because agents are collaborating and assisting each other (Fig. 1). The resulting solution must tolerate partitions to the network or long delays before data can be sent between nodes or back to a data center.

In this article, we formalize the *communication-aware computation task-scheduling problem* and present an integer linear

Manuscript received 26 October 2020; revised 26 May 2021, 17 October 2021, 14 December 2021, and 1 March 2022; accepted 1 May 2022. Date of publication 16 August 2022; date of current version 13 March 2023. This work was supported by National Aeronautics and Space Administration. Recommended by Associate Editor K. Savla. *(Corresponding author: Joshua Vander Hook.)*

The authors are with the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91101 USA (e-mail: hook@jpl.nasa.gov; federico.rossi@jpl.nasa.gov; tiago.stegun.vaquero@jpl.nasa.gov; martina.i.troesch@jpl.nasa.gov; marc.sanchez.net@jpl.nasa.gov; joshua.schoolcraft@jpl.nasa.gov; jean-pierre.de.la.croix@jpl.nasa.gov; steve.a.chien@jpl.nasa.gov).

Digital Object Identifier 10.1109/TCNS.2022.3198789

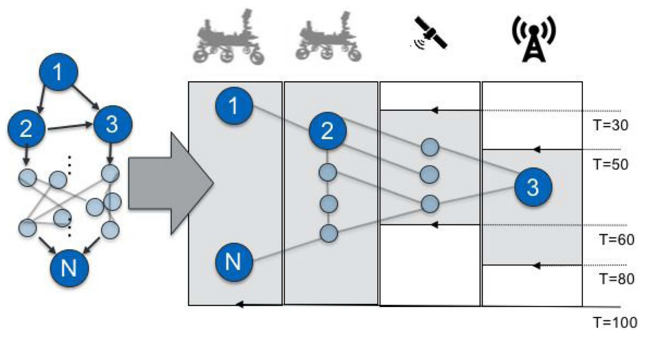

Fig. 1. Illustrative MOSAIC scenario. A set of processing tasks (on the left as dependency graph) must be mapped to multiple assets with heterogeneous computing, communication, and energy capacities. Each asset is also available over a fixed time window due to terrain effects or orbital parameters. The goal is to compute all the required tasks as quickly as possible.

program (ILP) that optimizes the allocation of computation and communication tasks to heterogeneous agents, accounting for the computational capabilities and time-varying communication links. Because data and computation are shared among many devices, we dub the resulting local computation-sharing network a Multirobot Onsite Shared Analytics Information and Computing (MOSAIC)) network.

We model and test with networks that use delay- and disruption-tolerant networking (DTN) which provides transparent store-and-forward, multihop data routing between arbitrary endpoints and negotiates intermittent interruptions and delays in connectivity [17], [18]. Unlike mobile phone networks which respond to arbitrary consumer actions, in cooperative multiagent networks, agents can explicitly share their goals and constraints with each other. Thus, we consider the robots' intended actions as part of the scheduling problem so that the robots can schedule data-intensive tasks when assistance is available. Our evaluation scenarios are biased toward multirover systems for Mars or the Moon. However, the results generalize to arbitrary time-varying communication graphs, such as vehicles on known street routes or constellations in orbit. In a multirover scenario, we show that distributed computation can increase the amount of science performed *three-fold* compared to the same system with no computational load sharing . We show that the solution includes intuitive results such as designated relay nodes and "assembly line" behaviors.

## A. Related Work

The core computational problem addressed in this work is communication-aware task scheduling. Task scheduling is known to be NP-complete [19]; furthermore, while polynomial-time approximation schemes for the problem exist, to the best of the authors' knowledge, no such schemes are known for the task-scheduling problem when computing nodes have *heterogeneous computational capabilities*, i.e., the same task requires different computation runtimes on different nodes [20]. A large number of heuristic algorithms have been proposed to solve the task-scheduling problem. Heuristics may be classified as *list-scheduling* heuristics (e.g., [21]), which rely on greedily allocating tasks according to a heuristic priority task assignment; *clustering* heuristics (e.g., [22]), which identify groups of tasks that should be scheduled on the same computing node; and *task duplication* heuristics, which duplicate some tasks to reduce communication overhead (e.g., [23]). In addition, a number of *guided random search* algorithms are available, including genetic algorithms [24] and ant colony optimization algorithms [25]. See the survey in [26] and introduction in [27] for a thorough review.

In particular, the heterogeneous earliest-finish-time (HEFT) heuristic algorithm [27] provides excellent performance for heterogeneous task-scheduling problems, and a number of variations of HEFT have been proposed [28], [29], [30]. However, the HEFT algorithm and its derivatives generally assume that computation nodes are able to perform all-to-all communication and that the availability of communication links does not change with time; they also do not capture access contention or bandwidth constraints on communication links, and do not accommodate *optional* tasks which are not required to be scheduled but result in a reward when added to the schedule.

Heuristic approaches are also used in model-based schedulers/temporal-planners that rely on activity-centric representations such as timeline-based modeling languages [31] and the Planning Domain Definition Language (PDDL) [32], [33], [34]. Research on PDDL temporal planners, for example, has focused on domain-independent heuristics and has deployed planning systems to several robotics applications, especially those that require both planning and scheduling capabilities [35], [36]. One of the main state-of-the-art temporal planners is OPTIC [37]; OPTIC not only reasons about actions' preconditions and effects to determine the set of actions required to achieve a given goal state, but also considers an action's temporal and resource constraints as well as soft state constraints (preferences) and continuous objective functions. Due to its generality in the input representation, we compare the performance of our approach with the OPTIC planner in Section IV-B.

Several heuristics are also available for the *online* scheduling problem, where computational tasks appear according to a stochastic process, and are not revealed to the scheduler in advance [38], [39], [40]; recent work extends such schedulers to accommodate communication latency constraints [41]. However, online approaches generally perform poorly compared to offline algorithms when the list of tasks to be executed is known in advance or in batch, a typical scenario for robotic exploration missions; in addition, even state-of-the-art online algorithms assume that *all-to-all communication between the computation nodes is available*. In contrast, the approach proposed in this article does adapt to realistic time-varying communication constraints, explicitly represents multihop communications between nodes, and accommodates optional tasks, while offering sufficiently fast computation times to make the approach amenable for field use, as we show.

The problem of resource-aware scheduling in space application has seen a significant amount of interest in the adaptive space systems community. However, existing solutions tend to focus on reconfigurability *within* an individual vehicle (see e.g. [42]);

solutions applicable to multiagent systems generally assume that all-to-all communication is available [43].

### B. Contribution

Our contribution is three-fold. First, we design a task-scheduling and task allocation algorithm based on integer programming that accounts for time-varying, bandwidth-constrained, multihop communication links and optional tasks, and that returns high-quality solutions quickly. We also provide a distributed implementation of the algorithm based on a shared-world, consensus-backed model. Second, we validate the performance of the algorithm with extensive benchmarking on several hardware architectures, including embedded architectures such as PPC 750 and Qualcomm Flight, and with human-in-the-loop field tests. Third, we explore and highlight emergent load sharing behaviors produced by the scheduling algorithm, and we quantitatively show that sharing of computational tasks can result in significant increases in science throughput for a notional multirobot mission. Finally, we provide an open-source implementation of the core results for the community's use.

Collectively, the results in this article show that sharing of computational tasks among heterogeneous agents greatly enhances heterogeneous multiagent architectures, resulting in higher utilization of computational resources, lower energy use, and increased scientific throughput for a given hardware architecture.

A preliminary version of this article was presented at the 2019 International Conference on Automated Planning and Scheduling (ICAPS) [15]. In this extended version, we i) provide an in-depth discussion of the ILP problem and several additional extensions (including additional cost functions and first-order modeling of network interference), ii) rigorously show that a flooding-based algorithm can be used to provide a distributed implementation of the scheduling algorithm for systems with moderate numbers of agents, iii) report extensive benchmarking results showing that the ILP can be solved effectively on embedded hardware architectures suitable for robotic systems, and iv) present an extended discussion of experimental results.

### C. Organization

The rest of this article is organized as follows. In Section II, we rigorously describe the multirobot, communication-aware computation task-scheduling problem solved in the article. In Section III, we provide a detailed description of the proposed scheduling algorithm. In Section IV, we present experimental results from a field test performed at Jet Propulsion Laboratory (JPL) and highlight a number of interesting emerging organization behaviors. We also report benchmarks showing that the proposed scheduling algorithm performs well on several embedded hardware architectures. Finally, Section V concludes this article.

## II. Problem Description

We now describe the communication-aware computation task-scheduling problem for heterogeneous multirobot systems.

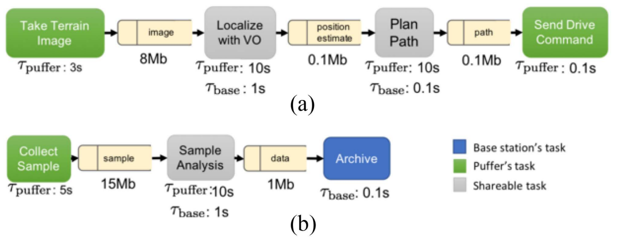

Fig. 2. Notional software network for the PUFFERs rovers.

### A. Tasks and Software Network

We wish to schedule a set $\mathbb{T}$ of tasks, with $|\mathbb{T}| = M$ (that is, the number of tasks in $\mathbb{T}$ is $M$). Computational tasks of interest can include, e.g., localizing a robot, computing a motion plan for a robot, classifying and labeling the content of an image [44], [45], or estimating the spatial distribution of a phenomenon based on point measurements from multiple assets [46], [47].

Tasks may be *required* or *optional*. Required tasks, denoted as $\mathbb{R} \subseteq \mathbb{T}$, must be included in the schedule. Optional tasks $\mathbb{T} \setminus \mathbb{R}$ are each assigned a *reward* score, denoted as $r(T)$ for each optional task $T \in \mathbb{T} \setminus \mathbb{R}$, which captures the value of including the task in a schedule.

The output of each task is a *data product*. Data products for task $T$ are denoted as $d(T)$. The size (in bits) of the data products are known a priori as $s(T)$ for task $T$.

Tasks are connected by dependency relations encoded in a *software network* $SN$. Let $P_T \subset \mathbb{T}$ be a set of predecessor tasks for task $T \in \mathbb{T}$. If task $\hat{T} \in P_T$ (that is, $\hat{T}$ is a *predecessor* of task $T$), task $T$ can only be executed by a robot if the robot has data product $d(\hat{T})$. If $\hat{T}$ is scheduled to be executed on the same robot as $T$, $d(\hat{T})$ is assumed to be available to $T$ as soon as the computation of $\hat{T}$ is concluded. If $\hat{T}$ and $T$ are scheduled on different robots, $d(\hat{T})$ must be transmitted from the robot executing $\hat{T}$ to the robot executing $T$ before execution of $T$ can commence. An example of $SN$ used in our experiments is shown in Fig. 2.

To ensure a solution exists, we require two assumptions.

***Assumption 1 (Feasibility):*** There exists a schedule where all required tasks are scheduled.

***Assumption 2 (No circular dependencies):*** The software network $SN$ does not have cycles.

***1) Agents:*** Agents in the network represent computing units. Let there be $N \in \mathbb{Z}^+$ agents in the network. The agents are denoted by $A_1, A_2, \ldots, A_N$. Each agent has known on-board processing and storage capabilities.

The time and energy cost required to perform a task $T$ on agent $A_i$ are assumed to be known and denoted, respectively, as $\tau_i(T)$ and $C_i^e(T)$. Depending on the application, time and energy cost can capture the worst-case, expected, or bounded computation time and energy; they are all considered to be deterministic.

***2) Contact Graph:*** Agents can communicate according to a prescribed time-varying *contact graph* $CG$, which denotes the availability and bandwidth of communication links between the robots.

$CG$ is a graph with time-varying edges. Nodes $\mathcal{V}$ in $CG$ correspond to agents. For each time instant $k$, directed edges $\mathcal{E}_k$ model

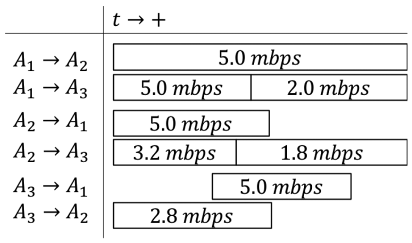

Fig. 3. Contact graph for three agents showing connectivity time windows and bandwidths available.

the availability of communication links; that is, $(i, j) \in \mathcal{E}_k$ if node $i$ can communicate to node $j$ at time $k$. Each edge has a (time-varying) data rate ranging from 0 (not connected) to $\infty$ (communicating to self), denoted by $r_{ij}(k)$ for the rate from $A_i$ to $A_j$ at time $k$. An example timeline representation for three agents with available bandwidths can be seen in Fig. 3.

A key feature of DTN-based networking is contact graph routing (CGR) [17], [48]. CGR takes into account predictable link schedules and bandwidth limits to automate data delivery and optimize the use of network resources. Accordingly, by incorporating DTN's store-forward mechanism into the scheduling problem, it is possible to use mobile agents as *robotic routers* to ferry data packets between agents that are not directly connected.

Communicating the data product $d(T)$ from $A_i$ to $A_j$ at time $k$ requires time

$$\tau_{ij}(T) = \min_{\tau} \left( \tau \text{ such that } \int_{\kappa=k}^{k+\tau} r_{ij}(\kappa) d\kappa \geq s(T) \right)$$

that is, $\tau_{ij}(T)$ is the shortest time required to transmit a total of $s(T)$ bits at an instantaneous data rate $r_{ij}(\cdot)$ starting at time $k$. If the data rate $r_{ij}(\cdot)$ is constant through the communication window and sufficiently long for the transmission to occur, the expression can be simplified to $\tau_{ij}(T) = s(T/)r_{ij}(k)$.

We model agents as single-threaded computers. If a robot actually has multiple processors, even of different types, these can be accommodated by modeling each processor as a computing agent, and connecting physically colocated processors with infinite-bandwidth, zero latency communication links.

***Assumption 3 (Computational resource availability):*** Agents can only perform a single task at any given time, including transmitting or receiving data products.

***Assumption 4 (Communication self-loops):*** Agents take 0 time to communicate the solution to themselves.

***3) Schedule:*** A schedule is (a) a mapping of tasks to agents and start-times, denoted as $\mathbb{S} : T \rightarrow (A_i, k)$, where $i \in [1, \ldots, N]$ and $k \geq 0$, and (b) a list of interagent communications $(A_i, A_j, d(T), k)$ denoting the transmission of $d(T)$ from $A_i$ to $A_j$ from time $k$ to time $k + \tau_{ij}(T)$ : $(\int_k^{k+\tau_{ij}(T)} r_{ij}(\kappa) d\kappa = s(T))$.

***4) Optimization Objectives:*** We consider several optimization objectives (formalized in the following section), including

1) *Optional tasks:* maximize the sum of the rewards $r(T)$ for optional tasks $T$ that are included in the schedule;
2) *Makespan:* minimize the maximum completion time of all scheduled tasks;
3) *Energy cost:* minimize the sum of the energy costs $C_i^e(T)$ for tasks included in the schedule;

***5) Scheduling problem:*** We are now in a position to state the communication-aware computation task-scheduling problem for heterogeneous multirobot systems.

***Problem 1 (Communication-Aware Computation Task-Scheduling Problem for Heterogeneous Multirobot Systems).*** Given a set of tasks modeled as a software network $SN$, a list of computational agents $A_i$, $i \in [1 \ldots N]$, a contact graph $CG$, and a maximum schedule length $C^\star$, find a schedule that satisfies

1) the maximum overall computation time is no more than $C^\star$;
2) all required tasks $T \in \mathbb{R}$ are scheduled;
3) a task $T$ is only scheduled on agent $A_i$ at time $k$ if the agent has received all the data product $d(\hat{T})$ for predecessor tasks $\hat{T} \in P_T$;
4) every agent performs at most one task (including transmitting and receiving data products) at any time;
5) the selected optimization objective is maximized.

## B. Notes on Problem Assumptions

The assumption that a feasible schedule including all required tasks exists (Assumption 1) is appropriate for multirobot systems where each required task "belongs" to a specific robot (i.e., the task is performed with inputs collected by the robot, and the output of the task is to be consumed by the same robot). Examples of such tasks include localization, mapping, and path planning. In such a setting, it is reasonable to assume that each robot should be able to perform all of its own required tasks with no assistance from other computation nodes; on the other hand, cooperation between robots can decrease the makespan, reduce energy use, and enable the completion of optional tasks.

In most relevant space applications, e.g., surface-to-orbit communications, orbit-to-orbit communications, and surface-to-surface communication in unobstructed environments, the contact graph is predicted to a high degree of accuracy before the mission begins, and is updated regularly. In obstructed environments where communication models are highly uncertain (e.g., subsurface voids such as caves, mines, tunnels), a conservative estimate of the channel capacity could be used. Extending Problem 1 to explicitly capture uncertainty in the communication graph is an interesting direction for future research.

Finally, Problem 1 also assumes that the communication graph is not part of the optimization process. The problem of optimizing the contact graph by prescribing the agents' motion is beyond the scope of this article (for a good example of the vast literature, see [49], [50]); note the tools described in this article can be used as an optimization subroutine to numerically assess the effect of proposed changes in the contact graph on the performance of the multirobot system.

## III. Scheduling Algorithm

### A. Integer Linear Program (ILP)

We formulate Problem 1 as an ILP. We consider a discrete-time approximation of the problem with a time horizon of $C_d^\star$ time steps, each of duration $C^\star/C_d^\star$, corresponding to the maximum schedule length $C^\star$. As is common in ILP formulations, the number of time steps can be set to any value that balances runtime vs. granularity. The optimization variables are as follows.

- $X$, a set of Boolean variables of size $N \cdot M \cdot C_d^\star$. $X(i,T,k)$ is true if and only if agent $A_i$ starts computing task $T$ at time $k$.
- $D$, a set of Boolean variables of size $N \cdot M \cdot C_d^\star$. $D(i,T,k)$ is true if and only if agent $A_i$ has stored the data products $d(T)$ of task $T$ at time $k$.
- $C$, a set of Boolean variables of size $N^2 \cdot M \cdot C_d^\star$. $C(i,j,T,k)$ is true if and only if agent $A_i$ communicates part or all of data products $d(T)$ to agent $A_j$ at time $k$.

The optimization objective $R$ can be expressed as follows.

- Maximize the sum of the rewards for completed optional tasks

$$R_r = \sum_{i=1}^{N} \sum_{T\in\mathbb{T}\setminus\mathbb{R}} \sum_{k=1}^{C_d^\star-\tau_i(T)} r(T)X(i,T,k). \tag{1a}$$

- Minimize the makespan of the problem

$$R_M = -\max_{i\in[1,N]} \max_{T\in\mathbb{T}} \max_{k\in[1,C_d^\star]} (k+\tau_i(T))\,X(i,T,k). \tag{1b}$$

- Minimize the energy cost of the problem

$$R_e = -\sum_{i=1}^{N} \sum_{T\in\mathbb{T}} \sum_{k=1}^{C_d^\star} C_i^e(T)X(i,T,k). \tag{1c}$$

Therefore, Problem 1 is formulated as follows.

$$\underset{X,D,C}{\text{maximize}}\ R \tag{2a}$$

subject to

$$\sum_{i=1}^{N} \sum_{k=1}^{C_d^\star-\tau_i(T)} X(i,T,k) = 1 \quad \forall T\in\mathbb{R} \tag{2b}$$

$$\sum_{i=1}^{N} \sum_{k=1}^{C_d^\star-\tau_i(T)} X(i,T,k) \le 1 \quad \forall T\in\mathbb{T}\setminus\mathbb{R} \tag{2c}$$

$$X(i,T,k) \le D(i,L,k) \tag{2d}$$

$$\forall i\in[1,\ldots,N], T\in[1,\ldots,M], L\in P_T, k\in[1,\ldots,C_d^\star]$$

$$\sum_{T=1}^{M}\left[\sum_{j=1}^{N}(C(i,j,T,k)+C(j,i,T,k)) + \sum_{\hat{k}=\max(1,k-\tau_i(T))}^{k} X(i,T,\hat{k})\right]$$

$$\le 1 \quad \forall i\in[1,\ldots,N], k\in[1,\ldots,C_d^\star] \tag{2e}$$

$$D(i,T,k+1)-D(i,T,k) \le \sum_{\tau=1}^{k}\sum_{j=1}^{N}\frac{r_{ji}(\tau)}{s(T)}C(j,i,T,\tau) + \sum_{\tau=1}^{k-\tau_i(T)} X(i,T,\tau)$$

$$\forall i\in[1,\ldots,N], T\in[1,\ldots,M], k\in[1,\ldots,C_d^\star-1] \tag{2f}$$

$$C(i,j,T,k) \le D(i,T,k)$$

$$\forall i,j\in[1,\ldots,N], T\in[1,\ldots,M], k\in[1,\ldots,T] \tag{2g}$$

$$D(i,T,1) = 0 \quad \forall i\in[1,\ldots,N], T\in[1,\ldots,M]. \tag{2h}$$

Equation (2b) ensures that all required tasks are performed and (2c) ensures that optional tasks are performed at most once.

Equation (2d) requires that agents only start a task if they have access to the data products of all its predecessor tasks. Equation (2e) captures the agents' limited computation resources by enforcing Assumption 3. Equation (2f) ensures that agents learn the content of a task's data products only if they i) receive such information from other agents (possibly over multiple time steps, each carrying a fraction $r_{ij}(k)/s(T)$ of the data product) or ii) complete the task themselves. Equation (2g) ensures that agents only communicate a data product if they have stored the data product themselves. Finally, (2h) models the fact that data products are initially unknown to all agents.

The ILP has $N^2MC_d^\star + 2NMC_d^\star$ Boolean variables and $M(N(3C_d^\star-1)+N)+NC_d^\star$ constraints; instances with dozens of agents and tasks and horizons of 50–100 time steps can be readily solved by state-of-the-art ILP solvers, as shown in Section IV.

### B. Modeling Extension: Capturing Network Interference

The ILP formulation can be extended to capture network interference as follows. In (2), link bandwidths $r_{ij}$ are assumed to be fixed and independent of each other: that is, the communication bandwidth $r_{ij}$ on a link is assumed to be achievable regardless of communication activity on other links. This may not hold for systems with robots in close proximity that share the same wireless channel. In such a setting, interference introduces a coupling between the achievable bandwidths on different links, and the amount of data that can be exchanged by interfering links is limited by the *channel capacity* of the shared physical medium.

The formulation in (2) can be extended to capture a first-order approximation of this effect, letting individual link bit rates be decision variables subject to constraints on the overall channel capacity. Effectively, agents are allowed to use less than the full capacity of individual links to ensure that their transmissions do not cause interference on other links sharing the same wireless channel.

To accommodate, define an additional set of real-valued decision variables $R$ of size $N^2 \cdot M \cdot C_d^\star$. $R(i,j,T,k)$ denotes the amount of bits of the data product of task $T$ that is transmitted from agent $A_i$ to agent $A_j$ in time interval $k$.

Under this model, the interfering links' channel capacity $r(I,k)$ (that is, the overall amount of bits that links in $I$ can

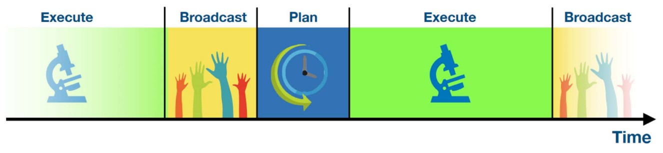

Fig. 4. Distributed implementation of the ILP relies on a broadcast–plan–execute cycle. First, agents exchange information about their own state through a message-passing algorithm and achieve a consensus on the system state. Next, all agents solve Problem (2) with the system state as input and with a deterministic stopping criterion. Finally, all agents execute the tasks assigned to them by the solution to Problem (2).

simultaneously transmit) is known, for each discrete time interval $k$ and each subset $I \in \mathbb{I} \subset 2^{N^2}$ of links that is subject to mutual interference. In order to avoid introducing an exponential number of constraints, it is desirable to consider a modest number of sets of interfering links based on the agents' geographical proximity. For instance, if all robots are operating in close proximity and can interfere with each other, the overall bandwidth of *all* links should be constrained to be smaller than the capacity of the shared channel, resulting in the addition of a single interference constraint.

Equation (2f) is replaced by the following equations:

$$R(i,j,T,k) \le r_{i,j}(k)C(i,j,T,k)$$
$$\forall i,j \in [1,\dots,N], T \in [1,\dots,M], k \in [1,\dots,T] \tag{3a}$$
$$D(i,T,k+1) - D(i,T,k)$$
$$\le \sum_{\tau=1}^{k}\sum_{j=1}^{N}\frac{1}{s(T)}R(j,i,T,\tau) + \sum_{\tau=1}^{k-\tau_i(T)} X(i,T,\tau)$$
$$\forall i \in [1,\dots,N], T \in [1,\dots,M], k \in [1,\dots,C_d^\star - 1] \tag{3b}$$
$$\sum_{i,j\in i}\sum_{T\in[1,\dots,M]} R(j,i,T,k) \le r(i,k) \quad \forall i \in I, k \in [1,\dots,T]. \tag{3c}$$

Equation (3a) ensures that the effective bit rate on a link is nonzero only if a communication occurs on the link; (3b) models the process by which robots learn data products through communication, closely following (2f); and (3c) ensures that the sum of all effective bit rates on interfering links does not exceed the channel capacity.

## C. Distributed, Real-Time Implementation

In order to provide a *distributed*, *real-time* implementation of the scheduler presented above suitable for field use, we leverage a shared-world approach using a "broadcast, plan, and execute" cycle (shown in Fig. 4).

Agents are assumed to have access to a common clock and have preexisting knowledge of the duration of the broadcast, plan, and execute phases of the cycle. The agents also know what programs or processes may be included in the software network $SN$(even if not all agents can execute all processes). They are not aware of the optimization objective, namely, the execution times, energy costs, sequences, and rewards of individual tasks.

***1) Broadcast:*** At an agreed-upon time, agents start the "broadcast" phase; during this phase, agents exchange their state with all other agents through a flooding message-passing algorithm [51, Ch. 4], and achieve a consensus on the overall system state. The duration of the broadcast phase is selected to ensure that consensus can be achieved for any possible network topology. As discussed in the Extended Version [52], if the communication network is strongly connected, systems with 10–50 agents can achieve consensus in under a second under conservative assumptions on the size of agents' states and available link bandwidths.

The state of each agent includes i) the estimated present and future bandwidths $r_{ij}$ between each agent and their neighbors, ii) the time and energy costs $\{\tau_i(T)\}_{i\in[1,N],T\in\mathbb{T}}$, $\{C_i^e(T)\}_{i\in[1,N],T\in\mathbb{T}}$ required by the agent to perform each possible task, and iii) the rewards $\{r(T)\}_{T\in\mathbb{T}\setminus\mathbb{R}}$ for performing optional tasks.

This approach is responsive to time-varying task rewards and agent capabilities. However, the choice of a single broadcast epoch per round does cause some delay in responsiveness, since agent capabilities and rewards can only be updated if they appear prior to the start of the broadcast phase for each cycle.

***a) Plan:*** Once the broadcast phase is over, agents switch to the "plan" phase. In this phase, each agent solves Problem (2) with the network topology, tasks set, and vehicle states computed in the broadcast phase as inputs.

Problem 1 is in general NP-hard, and a solver may fail to find an optimal solution within the allocated time. To ensure that a feasible final solution is found, we provide the solver with a trivial initial solution (which exists, according to Assumption 1). To ensure that all agents agree on the same solution, we use a *deterministic* MILP solver (i.e., a solver that explores the decision tree according to a deterministic policy), and we employ a deterministic stopping criterion (i.e., the solver terminates after a prescribed, deterministic number of branch-and-bound steps, selected to ensure termination within the duration of the "plan" phase).

***b) Execute:*** Once the plan phase is over, agents switch to the execution phase; here, each agent reads the output of Problem (2) and executes the tasks that are assigned to itself according to the timing prescribed by the schedule.

This approach provides a *distributed* and *anytime* implementation of Problem 1 which we implement and test in the next section.

## D. Remarks

In the problem formulation, communication tasks do consume computational resources on both the transmitter and the receiver (Assumption 3). This is in line with the current mode of operations of space missions, where communication is not concurrent with other activities due to computational, power, and reliability considerations. As a result, the rovers' activities should be *synchronized:* In absence of synchronization, a rover's transmission could interrupt computational activities on the receiver, or be lost if the receiver is unavailable.

In the light of this, the selection of a "broadcast–plan–execute" distributed implementation, which relies on the availability of synchronization between the agents, is preferable for its simplicity and ease of verification.

In cases where agents can concurrently communicate and perform computational tasks, a more versatile *asynchronous* distributed implementation could also be used. We propose such an asynchronous load sharing execution mechanism in [53]; the proposed architecture is agnostic to the task-allocation mechanism used, and is therefore compatible with plans provided by the ILP.

The broadcast–plan–execute cycle relies on synchronization of the agents' clocks and on accurate knowledge of the duration of tasks to be executed. Deviations from predicted execution times can result both in tasks not being completed in the "execute" phase, and in missed communication windows (if, e.g., a task is not completed by the time its data products should be transmitted to another agent). The cyclic nature of the broadcast–plan–execute cycle allows "missed" tasks to be rescheduled at a later time step; nevertheless, the design of *robust* scheduling algorithms that can accommodate uncertainty in synchronization and in task execution times is an interesting direction for future research.

While the flooding-based synchronization mechanism itself is quite robust (as discussed in the previous subsection), the overall scheduling approach is *not* robust to failures of the broadcasting synchronization mechanism. The integration of more robust coordination mechanisms (e.g., challenge–response to verify that agents have achieved a consensus, and watchdogs triggering the execution of an agreed-upon contingency plan) is an interesting direction for future research.

Finally, the complexity of the ILP scales exponentially with the number of agents; accordingly, in principle, it may be infeasible to obtain a high-quality solution to Problem 2 at a sufficient cadence for control of a multirobot system, especially on embedded platforms. However, in Section IV-B, we show that state-of-the-art ILP solvers can provide high-quality (if not optimal) solutions within tens of seconds, even on highly constrained platforms.

## IV. Experiments

In this section, we explore the performance of the proposed approach on a variety of realistic problems. First, we present field tests of a *distributed* implementation on a set of mobile, wirelessly connected, Raspberry Pis. Second, we assess the computational complexity and performance of the approach through rigorous benchmarks on a variety of computational architectures.

### A. Distributed Hybrid Implementation

The goals of these experiments were to implement and test the *distributed* version of Problem (2), and, specifically, the broadcast–plan–execute architecture in Section III-C, in realistic and challenging environments. We sought to check for four important characteristics of a field-deployed system.

- *Robustness:* Does the proposed approach run for extended periods of time? It may crash, we may encounter violated assumptions, or the ILP may not find a feasible solution in time.
- *Computational cost:* Does the implementation scale well and run quickly on realistic computing architectures?
- *Networking:* Are communication tasks scheduled reasonably, despite the additional complexity of scheduling computation? Since our ILP contains data routing as a subproblem, we expect the solutions to contain reasonable routing behaviors.
- *Load Balancing:* Does the solution exhibit load-balancing behaviors when nodes with uneven computational load have good communication between them?
- *Science Optimization:* Does the system achieve an increase in throughput of science data compared to a naive approach?

We used a notional multirobot scenario where multiple small rovers perform both "housekeeping" tasks (e.g., sensing, path planning) and science tasks (e.g., microscope measurements) and are aided by a computationally capable base station. This is illustrated in Fig. 5. The concept of operations is loosely based on JPL's PUFFER robots [54].

The software network used is shown in Fig. 2. Tasks are arranged in two sets. "Housekeeping" tasks (Fig. 2, top) are based on the Mars Perseverance rover's autonomy architecture, and their execution time is based on actual benchmarks on Perseverance's on-board RAD750 [55]. Housekeeping tasks include i) capturing an image of the terrain, ii) self-localization based on that image, iii) planning a path through the environment, and iv) dispatching the drive command. While image capture and drive command have to be executed on board, localization and path planning tasks can be delegated to another robot in the network.

To model optional, autonomous science activities, we also added the "science tasks" shown in the bottom of Fig. 2. Specifically, PUFFERs can i) collect a sample from the environment; ii) analyze it; and iii) send the analysis data to the base station for storage and eventual uplink. The sample analysis task can be assigned to another node. Only agents inside predesignated "science zones" can perform sample collection; storage must be performed by the base station. Each science task has a reward associated to it; the reward for sample collection is set to 5, the reward for data analysis is 10, and the reward for storing data is 20. Note that no actual sampling and analysis tasks were executed; rather, task execution was simulated by allocating time in the schedule computed by each node.

This set of "science tasks" represents the scenario where PUFFERs explore a distributed but spatially correlated phenomenon, such as water moisture levels, by performing kriging [47], a process routinely used for spatial estimation in farming on Earth [46].

The base station's computational power is an order of magnitude larger than an individual robot's, and it is equipped with the same communication equipment as the other nodes in the network. The base station is not assigned any required tasks; its key role is to serve as a supporting node for sharing the computational load of the network.

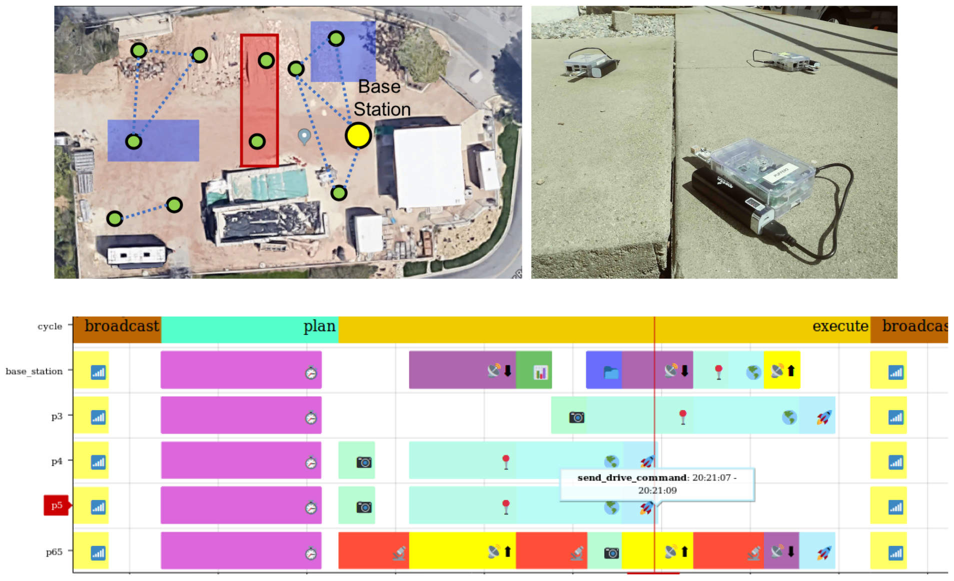

Fig. 5. Illustrative scenario in the Mars Yard at JPL (top left), pictures of the hardware nodes (top right), and one scheduled timeline (bottom). Timelines represent the operational cycle and the task allocation. Communication links can be disabled to test system adaptation and relocation of tasks. RVIZ view provides vehicle positioning and network topology information.

In the field experiments, the PUFFERs were represented by Raspberry Pis (model 3) with a GPS receiver, and the base station was a desktop computer at a fixed location. The Pis were moved about within an outdoors experimental area with two marked "science zones" by human experimenters. We had limited control over positions of the nodes during the experiment and demonstration, due to the use of the highly portable Raspberry Pis and participation of enthusiastic observers from JPL and direction from observing sponsors. Accordingly, this experiment was an ideal test of the *reliability* and *robustness* of the overall architecture; separate software benchmarks (reported in the next section) are better suited to assess the computational cost and performance of the approach.

To control the communications network, all platforms were connected through a WiFi router; bandwidths between nodes were computed in simulation based on i) interagent distance; ii) the presence of no-communication zones (shown in red in Fig. 5); and iii) direct human intervention (i.e., the experimenters' ability to disable selected communication links on demand). Data rates between agents were based on interagent range according to a piecewise constant function: Available bandwidth ranged from 1 Mbps for ranges between 15 and 200 m to 11 Mbps for ranges between 0 and 5 m, or zero (unavailable) if the line-of-sight between two vehicles crossed obstructions (shown as red zones in Fig. 5).

The logical layers on all agents was implemented using the Robot Operating System (ROS). For each vehicle in the network, we developed a set of simulated components, including a communication system to handle the broadcasting phase and transfer of data, a science component to simulate science tasks, a navigation/localization system to keep track of the vehicle's pose, and a controller to plan and dispatch tasks according to the scheduler.

We also developed tools to visualize 1) the execution timeline (and task allocation); 2) the network topology and bandwidth; and 3) the vehicles' positions in the environment. The tools also enable manipulation of the simulation by adding additional no-communication zones and removing communication links. The software interface is shown in Fig. 5.

The agents were assumed to have access to a common clock, and we considered an operational cycle 45 s long, where 5 s was dedicated to the "broadcast" phase, 10 s the "plan" phase, and 30 s to execution (in line with the expected duration of the tasks in the software network in Fig. 2). In practice, we would expect a much longer execution cycle, and perhaps shorter broadcast/plan phase, but these values allowed increased introspection and iteration during field tests.

The ILOG CPLEX solver was used to solve the ILP. Since CPLEX does not support an ARM architecture, we deployed the scheduler in an x86 server (AWS m5.xlarge) that was queried at each plan phase by each vehicle. Vehicles called the scheduler independently; the solution received by each vehicle was guaranteed to be consistent with the other vehicles' through use of a deterministic solver with a deterministic stopping criterion (as discussed in Section III-C). The deterministic amount of solver steps corresponded to approximately 10 s of execution on the Pi. To ensure anytime availability of a feasible solution, the solver was seeded with an initial solution where agents did not share any computational tasks and executed no optional tasks, which is guaranteed to be feasible.

Representative portions of the field test are shown in a video in the Supplementary Material.[1] We provide a software release for full exploration of the results (Section IV-B). Experiments were run for four hours, and demonstrated all of the following characteristics.

***1) Robustness:*** During the 4-h-long demonstration, nodes were added and removed from the network (by activating and

[1]Available at https://youtu.be/zTQ7Y4-ax2A.

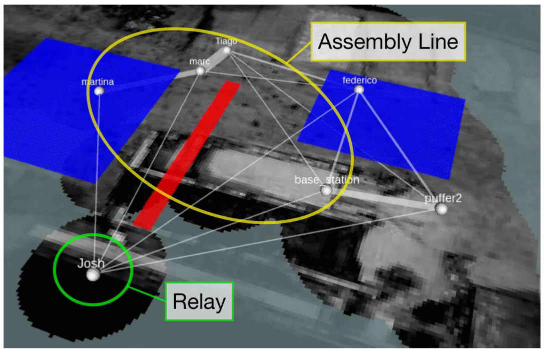

Fig. 6. Relay and assembly line emerging behaviors (yellow and green annotations were added manually to the RViz output from the field demonstration).

deactivating the corresponding Raspberry Pi's), and active nodes were moved around by observers, including in and out of science zones. We verified that the proposed approach is able to *consistently* provide good solutions to problems with 3 to 15 nodes within the 10-s planning window, and that the broadcast–plan–execute architecture can be used to provide a distributed implementation of Problem (2) that is robust to unforeseen, human-driven changes in the network topology and in the tasks to be scheduled.

***a) Networking and Data Relay:*** One of the most intuitive, and obviously beneficial emergent behaviors we observed was that of relay activities. Relay nodes, informally speaking, did nothing more than relay communications between other nodes while tending to their own housekeeping tasks. This behavior was induced reliably through the use of the no-communication zone to block direct communication with the base station. Traffic was reliably routed though nodes that were between the base station and the sender instead, as shown in Fig. 6.

***b) Load Balancing and Science Clusters:*** The choice of the software network places additional load on nodes that are in "science zones," by adding (optional) science tasks to their list of tasks. In a recurring behavior, "science clusters" formed whenever one vehicle was inside a science region, and other vehicles were nearby but outside. For example, in Fig. 6, the nodes in both science zones off-loaded their localization and path-planning tasks to the other nearby agent, so as to perform multiple sample collection and analysis tasks.

***c) Science Optimization:*** Due to the timing chosen for the software network, the proposed approach could yield at most a threefold increase in the number of optional science tasks performed for each node in a science zone. That is, the sum of the computation times of all relocatable housekeeping tasks was twice the cost of a science task; therefore, by doing *only* science and offloading all relocatable housekeeping tasks, an agent could gather three times more science than would have been possible with no load sharing. The additional analysis and storage tasks placed additional load on nodes *outside* of the science zone, if appropriately tasked. This threshold was achieved for some nodes that had sufficient nearby nodes, and sufficient throughput to the base station. Again in Fig. 6, the left science node was able to schedule three sample-gather tasks, by offloading tasks to nearby agents. We can explore the likelihood of this occurring in random networks in Section IV-B.

***d) Science Optimization With Assembly Lines:*** The combination of relay and load balancing produces and interesting result that was unintended but obvious in hindsight. When the system did reach the maximum observed science throughput, the relay nodes also served as computational aids for the science tasks, analyzing the data enroute to the base station akin to an "assembly line." We illustrate an occurrence of such a case in Fig. 7. The node labelled PUFFER 1 is in the left-most science zone and offloads localization (cyan) to nearby PUFFER 6, as in the "science cluster" scenario. PUFFER 1 also schedules three samples (red). Two sample data products are then transferred to PUFFER 2, which acts as a relay to the base station. PUFFER 2 analyzes one sample and forwards the resulting analysis result and one sample data product to PUFFER 3; PUFFER 3 analyzes the sample and transfers two analyzed data products to the base station for storage. The third sample data product is not analyzed or stored due to the short time horizon; nevertheless, it is collected to receive the corresponding reward. As mentioned, a threefold increase per node in science zones is the maximum possible increase due to the amount of time to execute compared to the time costs of all the tasks.

The "assembly line" result is quite interesting and may have unexplored efficiency increases for edge computing networks like terrestrial 5G networks.

***e) Store and Forward, and Data Muling:*** Because the planner has knowledge of the future state of the communications network, it should be possible to plan for future connectivity and store-and-forward packets to a node in preparation for a link coming online. If the link comes online because the storing node *moves*, this is sometimes called "data muling" [56]. We did not observe this in field testing because we could not predict the future state of the communications network, due to human manipulation. However, the data muling behavior was readily observed and reproduced in simulation, as shown in Fig. 8 and in the video in the Supplementary Material.

## *B. Software Benchmarks*

Next, we show through numerical results that the proposed ILP can be solved efficiently on a variety of hardware platforms, including embedded platforms suitable for robotics applications, and we explore the benefits of the approach compared to a "selfish" scenario, where agents cannot share computational tasks. To this end, we test the performance of a *centralized* version of Problem (2) on several hardware architectures for 20 randomly generated network topologies (shown in Fig. 9) and several cost functions. In each scenario, a subset of the agents was randomly placed in "science zones"; agents in science zones were able to collect one sample, which could optionally be analyzed and stored.

For each instance, the number of agents (proportional to number of tasks to schedule) was varied from 2 to 13 agents to assess the scalability of the proposed approach. Optimization

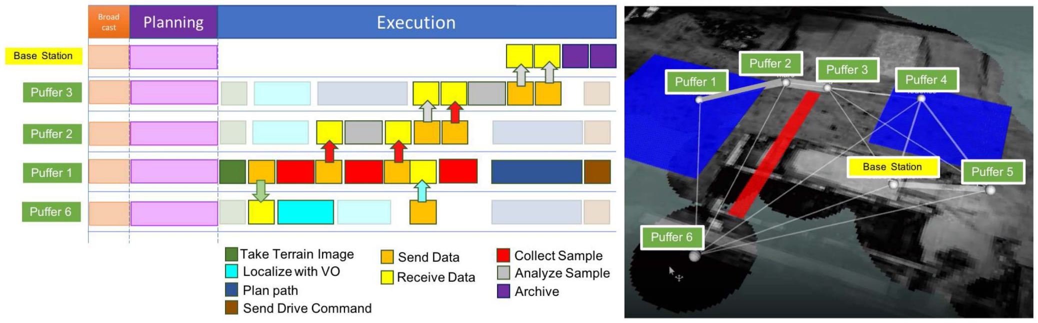

Fig. 7. Illustrative example of the assembly line case.

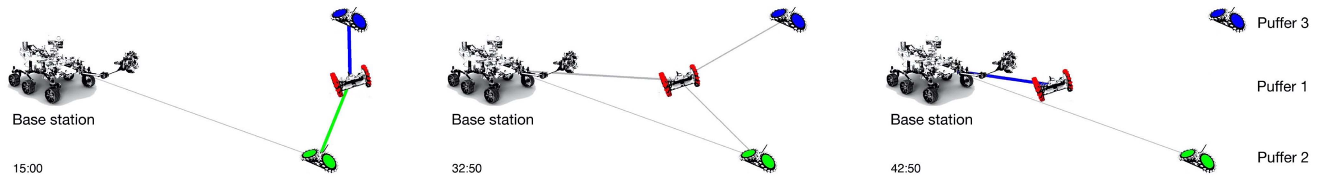

Fig. 8. Example simulation of the data mule scenario. Left: Three Puffers have a weak link to the base station, but the middle robot will move closer and so both robots transmit their data to it rather than directly to the base station. Right: Later, the red robot transmits all data to the base station. See Supplementary videos and Section IV-B.

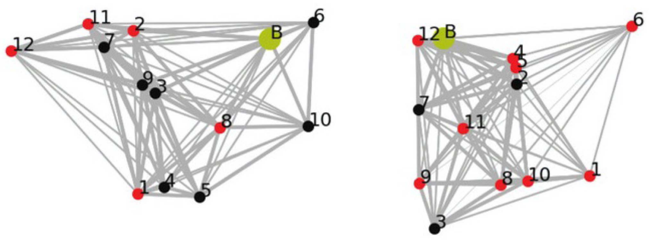

Fig. 9. Two example scenarios from the numerical experiments. The base station is yellow. Nodes able to perform science tasks are in red; nodes unable to perform science tasks are shown in black, and the width of edges shows bandwidth.

objectives included i) maximization of reward from optional task; ii) minimization of energy expenditure; and iii) a linear combination of the two. The problem was solved on several computing platforms, specifically

1) a modern Intel Xeon workstation equipped with a 10-core E5-2687 W processor;
2) an embedded Qualcomm Flight platform equipped with a APQ8096 SoC;
3) a Powerbook G3 computer equipped with a single-core PowerPC 750 clocked at 500 MHz, the same CPU (albeit without radiation tolerance adjustments) as the RAD750 used on the Curiosity and Mars 2020 rovers [57].

The ILP was solved with the SCIP solver [58]. For each problem, we computed both the time required for the solver to find and certify an optimal solution, and the quality of the best solution obtained after 60 s of execution. We also compared the performance of the proposed scheduler with the state-of-the-art OPTIC PDDL scheduler [37]. Results are shown in Fig. 10.

On the Xeon architecture, the median solution time for problems with up to six agents is under 10 s, and the median solution time for problems with up to nine agents is under 100 s (Fig. 10, top). The proposed anytime implementation is consistently able to find an optimal solution for problems with up to 11 agents in under 60 s (Fig. 10, middle). On the embedded Qualcomm SoC, the median solution time for problems with up to four agents is under 10 s, and the anytime implementation finds the optimal solution to problems with up to eight agents in under 60 s. Finally, even the highly limited PPC 750 processor is able to find an optimal solution to problems with up to five agents in under 60 s—a remarkable achievement for a 20-year-old processor.

The ILP scheduler offers superior performance compared to the anytime implementation of the OPTIC scheduler (Fig. 10, bottom). In particular, solving Problem (2) results in higher quality solutions for a given problem size and execution time, and OPTIC is unable to return solutions for problems with more than seven agents even on the Xeon architecture.

We also assessed the potential benefits of the proposed approach on a more complex version of the problem, where each agent in a "science zone" was able to collect up to *three* samples. We solved the same set of problems shown in Fig. 9 with up to nine agents; for each instance, we compared the solution to the ILP with a "selfish" allocation where agents were not allowed to share computational tasks (except for the storage task, which was constrained to be executed on the base station). We evaluated the solution quality both after 60 s of execution, and after 3600 s of execution (a time sufficient to achieve and prove optimality for the vast majority of the scenarios considered).

Fig. 11 shows the overall number of tasks performed and the average energy usage per task across all problem instances. After 1 h of execution, the proposed approach yields a 37.3% increase

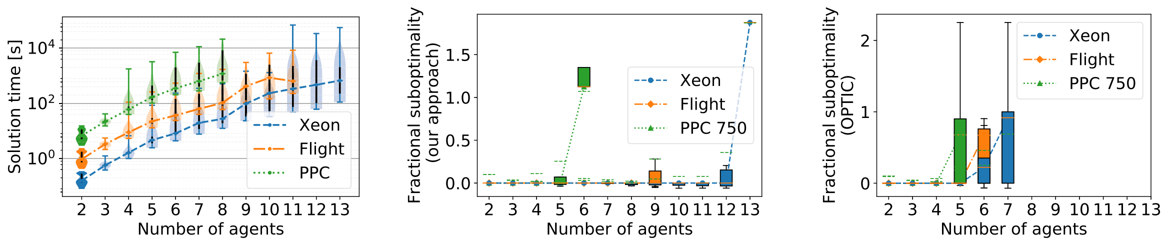

Fig. 10. Numerical results on several hardware platforms, and comparison with the OPTIC PDDL scheduler. Left: Time required to solve Problem 2 to optimality. Middle: Suboptimality as a fraction of the optimal solution after 60 s of execution for Problem (2) and (right) for the OPTIC PDDL solver [37].

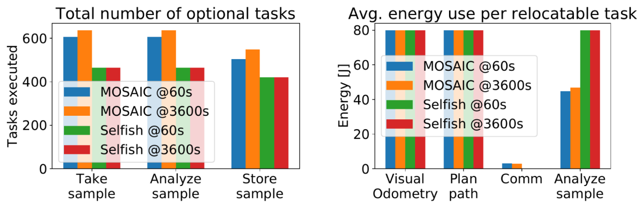

Fig. 11. Proposed approach vs. "selfish" approach (no sharing of CPU time). Left: tasks performed. Right: average energy per task.

in the number of samples collected and analyzed, and a 30.4% increase in the number of samples stored, compared to the selfish approach; the approach also results in a 41.4% reduction in average energy use for the sample analysis task, which more than outweighs the small increase in energy use for communications. Remarkably, a similar trend is observed even when the solver is stopped after 60 s: Here, the proposed approach results in a 30.6% increase in the number of samples collected and analyzed, a 19.7% increase in the number of samples stored, and a 44% decrease in the average energy use for sample analysis compared to the selfish approach.

Collectively, these results show that the proposed approach holds promise to yield significant increases in scientific returns and decreased energy usage; can be implemented on embedded robotic architectures with modest computational performance; and performs well in highly dynamic environments, making it well-suited for field robotics multiagent applications.

***1) Reproducing Our Results:*** We have released implementations of Problem (2) using the CPLEX, SCIP, and GLPK MILP solvers under a permissive open-source license. The implementations are available online at github.com/nasa/mosaic. Provided scenario files allow reproduction of all of the emergent behaviors discussed.

## V. Conclusion

In this article, we described the communication-aware computation task-scheduling problem for heterogeneous multirobot system and the MOSAIC architecture. We proposed an ILP formulation that allows to optimally schedule computational tasks in heterogeneous multirobot systems with time-varying communication links. We showed that the ILP formulation is amenable to a distributed implementation; can be solved efficiently on embedded computing architectures; and can result in a threefold increase in science returns compared to systems with no computational load sharing.

A number of directions for future research are of interest. First, we plan to explore pathways to infusion of the MOSAIC architecture in future multirobot planetary exploration missions. The proposed Mars sample return mission concepts plan to revisit the same area with multiple launches to fetch, retrieve, and eventually launch soil samples for return to Earth [59]. This offers an especially attractive avenue for deployment of MOSAIC, where each deployed asset could act as an "infrastructure upgrade," providing communication, computation, and data analysis services for all subsequent assets. Agents participating in the MOSAIC could include Cubesats similar to MarCO [60]; assets embedded in the "sky crane" lander and dropped during the "flyaway" phase [61]; tethered balloons [62]; and aerostationary orbiters providing constant assistance to half the Mars surface [63]. The algorithms proposed in this article can be used during the system design phase to optimize the hardware of the distributed missions by simulating the scheduling problem in the loop with an iterative hardware trade space explorer such as [64].

Second, we will design software libraries and middlewares that enable integration of the proposed scheduler with existing autonomy software, autonomously and transparently distributing computational tasks according to the optimal schedule. A preliminary effort in this direction can be found in [53].

Finally, it is of interest to extend the proposed scheduling approach to handle uncertainty in the contact graph and in the execution time of individual task. One promising research avenue is to incorporate stochastic optimization tools as well as probabilistic planning and scheduling approaches [65] to the computation sharing problem, which holds promise to provide guarantees that the MOSAIC is able to operate within given bounds on the uncertainty of the problem inputs.

## References

[1] E. Ackerman, "Robots conquer the underground: what darpa's subterranean challenge means for the future of autonomous robots," *IEEE Spectrum*, vol. 59, no. 5, pp. 30–37, 2022.
[2] J. Waterston, J. Rhea, S. Peterson, L. Bolick, J. Ayers, and J. Ellen, "Ocean of things: Affordable maritime sensors with scalable analysis," in *Proc. IEEE OCEANS Conf. Marseille*, 2019, pp. 1–6.
[3] H. J. Kramer and A. P. Cracknell, "An overview of small satellites in remote sensing," *Int. J. Remote Sens.*, vol. 29, no. 15, pp. 4285–4337, 2008.
[4] S. M. Pekkanen, "Governing the new space race," *AJIL Unbound*, vol. 113, pp. 92–97, 2019.

[5] R. J. Léveillé and S. Datta, "Lava tubes and basaltic caves as astrobiological targets on earth and mars: A review," *Planet. Space Sci.*, vol. 58, no. 4, pp. 592–598, 2010.

[6] A. S. McEwen et al., "Recurring slope lineae in equatorial regions of Mars," *Nature Geosci.*, vol. 7, no. 1, 2014, Art. no. 53.

[7] B. Balaram et al., "Mars helicopter technology demonstrator," in *Proc. AIAA Atmospheric Flight Mechanics Conf.*, 2018, Art. no. 23.

[8] I. A. Davydychev, J. T. Karras, and K. C. Carpenter, "Design of a two-wheeled rover with sprawl ability and metal brush traction," *J. Mechanisms Robot.*, vol. 11, no. 3, 2019, Art. no. 035002.

[9] R. Doyle et al., "High performance spaceflight computing (HPSC) next-generation space processor (NGSP): A joint investment of NASA and AFRL," in *Proc. Workshop Spacecraft Flight Softw.*, 2013, pp. 1–19.

[10] W. Powell et al., "Enabling future robotic missions with multicore processors," in *Proc. Infotech, Aerosp.*, 2011, Art. no. 1447.

[11] G. Mounce, J. Lyke, S. Horan, W. Powell, R. Doyle, and R. Some, "Chiplet based approach for heterogeneous processing and packaging architectures," in *Proc. IEEE Aerosp. Conf.*, 2016, pp. 1–12.

[12] A. G. Schmidt, G. Weisz, M. French, T. Flatley, and C. Y. Villalpando, "SpaceCubeX: A framework for evaluating hybrid multi-core cpu/fpga/dsp architectures," in *Proc. Aerosp. Conf.*, 2017 pp. 1–10.

[13] L. Turchi, S. Payler, F. Sauro, R. Pozzobon, M. Massironi, and L. Bessone, "A system for supporting distributed field science operations during astronaut training and planetary exploration: The electronic fieldbook," in *Proc. 52th Lunar Planet. Sci. Conf.*, 2021, Art. no. 1118.

[14] J. Vander Hook et al., "Dynamic shared computing resources for multi-robot mars exploration," in *Proc. 6th Workshop Plan. Robot., 28th Int. Conf. Automated Planning, Scheduling*, Delft, The Netherlands, 2018, pp. 11–18.

[15] J. Vander Hook et al., "Mars on-site shared analytics information and computing," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2019, pp. 707–715.

[16] E. Cuervo et al., "MAUI: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst., Appl., Serv.*, 2010, pp. 49–62.

[17] E. J. Wyatt et al., "New capabilities for deep space robotic exploration enabled by disruption tolerant networking," in *Proc. 6th Int. Conf. Space Mission Challenges Inf. Technol.*, 2017, pp. 1–6.

[18] S. Burleigh et al., "Delay-tolerant networking: An approach to interplanetary internet," *IEEE Commun. Mag.*, vol. 41, no. 6, pp. 128–136, Jun. 2003.

[19] M. R. Garey and D. S. Johnson, *Computers and Intractability*. San Francisco, CA, USA: W. H. Freeman, 1979.

[20] R. Graham, E. Lawler, J. Lenstra, and A. Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," in *Discrete Optimization II, Ser. Annals of Discrete Mathematics*, P. Hammer, E. Johnson, and B. Korte, Eds. Amsterdam, The Netherlands: Elsevier, 1979, vol. 5, pp. 287–326.

[21] G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 2, pp. 175–187, Feb. 1993.

[22] T. Yang and A. Gerasoulis, "DSC: Scheduling parallel tasks on an unbounded number of processors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 9, pp. 951–967, Sep. 1994.

[23] I. Ahmad and Y.-K. Kwok, "A new approach to scheduling parallel programs using task duplication," in *Proc. Int. Conf. Parallel Process.*, 1994, vol. 2, pp. 47–51.

[24] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Sci. Program.*, vol. 14, no. 3/4, pp. 217–230, 2006.

[25] W. Chen and J. Zhang, "An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements," *IEEE Trans. Syst., Man, Cybern., Part C (Appl. Rev.)*, vol. 39, no. 1, pp. 29–43, Jan. 2009.

[26] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surveys*, vol. 31, no. 4, pp. 406–471, Dec. 1999.

[27] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[28] X. Tang, K. Li, and D. Padua, "Communication contention in APN list scheduling algorithm," *Sci. China Ser. F: Inf. Sci.*, vol. 52, no. 1, pp. 59–69, Jan. 2009.

[29] L. Canon and E. Jeannot, "Evaluation and optimization of the robustness of DAG schedules in heterogeneous environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 4, pp. 532–546, Apr. 2010.

[30] X. Tang, K. Li, G. Liao, K. Fang, and F. Wu, "A stochastic scheduling algorithm for precedence constrained tasks on grid," *Future Gen. Comput. Syst.*, vol. 27, no. 8, pp. 1083–1091, 2011.

[31] S. Chien et al., "A generalized timeline representation, services, and interface for automating space mission operations," in *Proc. Int. Conf. Space Oper.*, 2012, Art. no. 1275459.

[32] M. Fox and D. Long, "PDDL2.1: An extension of PDDL for expressing temporal planning domains," *J. Artif. Intell. Res.*, vol. 20, pp. 61–124, 2003.

[33] S. Edelkamp and J. Hoffmann, "PDDL 2.2: The language for classical part of the 4th international planning competition," *Fachbereich Informatik Institut für Informatik*, Germany, Tech. Rep., 195, 2004.

[34] A. Gerevini and D. Long, "Preferences and soft constraints in PDDL3," in *Proc. ICAPS Workshop Plan. Preferences Soft Constraints*, 2006, pp. 46–53.

[35] M. Cashmore, M. Fox, T. Larkworthy, D. Long, and D. Magazzeni, "AUV mission control via temporal planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2014, pp. 6535–6541.

[36] A. Coles et al., "On-board planning for robotic space missions using temporal PDDL," in *Proc. Int. Workshop Plan. Scheduling Space*, 2019, pp. 1–9.

[37] J. Benton, A. Coles, and A. Coles, "Temporal planning with preferences and time-dependent continuous costs," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2012, pp. 2–10.

[38] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Trans. Autom. Control*, vol. 37, no. 12, pp. 1936–1948, Dec. 1992.

[39] J. G. Dai and W. Lin, "Maximum pressure policies in stochastic processing networks," *Oper. Res.*, vol. 53, no. 2, pp. 197–218, 2005.

[40] D. Terekhov, T. T. Tran, D. G. Down, and J. C. Beck, "Integrating queueing theory and scheduling for dynamic scheduling problems," *J. Artif. Intell. Res.*, vol. 50, no. 1, p. 535–572, 2014.

[41] C. Yang, A. S. Avestimehr, and R. Pedarsani, "Communication-aware scheduling of serial tasks for dispersed computing," in *Proc. IEEE Int. Symp. Inf. Theory*, 2018, pp. 1226–1230.

[42] M. Fayyaz, T. Vladimirova, and J.-M. Caujolle, "Adaptive middleware design for satellite fault-tolerant distributed computing," in *Proc. NASA/ESA Conf. Adaptive Hardware Syst.*, 2012, pp. 23–30.

[43] Y. Liao et al., "Caching-aided task offloading scheme for wireless body area networks with MEC," in *Proc. NASA/ESA Conf. Adaptive Hardware Syst.*, 2019, pp. 49–54.

[44] M. Ono et al., "Data-driven surface traversability analysis for MARS 2020 landing site selection," in *Proc. Aerosp. Conf.*, 2016, pp. 1–12.

[45] S. Higa et al., "Vision-based estimation of driving energy for planetary rovers using deep learning and terramechanics," *IEEE Robot. Automat. Lett.*, vol. 4, no. 4, pp. 3876–3883, Oct. 2019.

[46] P. Tokekar, J. V. Hook, D. Mulla, and V. Isler, "Sensor planning for a symbiotic UAV and UGV system for precision agriculture," *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1498–1511, Dec. 2016.

[47] A. Bárdossy and W. Lehmann, "Spatial distribution of soil moisture in a small catchment. Part 1: Geostatistical analysis," *J. Hydrol.*, vol. 206, no. 1, pp. 1–15, 1998.

[48] G. Araniti et al., "Contact graph routing in DTN space networks: Overview, enhancements and performance," *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 38–46, Mar. 2015.

[49] Y. Yan and Y. Mostofi, "Co-optimization of communication and motion planning of a robotic operation under resource constraints and in fading environments," *IEEE Trans. Wireless Commun.*, vol. 12, no. 4, pp. 1562–1572, Apr. 2013.

[50] A. Ghaffarkhah and Y. Mostofi, "Communication-aware motion planning in mobile networks," *IEEE Trans. Autom. Control*, vol. 56, no. 10, pp. 2478–2485, Oct. 2011.

[51] N. A. Lynch, *Distributed Algorithms*. Amsterdam, The Netherlands: Elsevier, 1996.

[52] J. Vander Hook et al., "Multi-robot on-site shared analytics information and computing," 2021, *arxiv.org/abs/2112.06879*.

[53] F. Rossi, T. Stegun Vaquero, M. Sanchez-Net, M. Saboia, and J. Vander Hook, "The pluggable distributed resource allocator (PDRA): A middleware for distributed computing in mobile robotic networks," in *Proc. IEEE/RSJ Int. Conf. on Intell. Robots Syst.*, Las Vegas, NV, USA, 2020, pp. 4337–4344.

[54] J. T. Karras et al., "Pop-up Mars rover with textile-enhanced rigid-flex PCB body," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 5459–5466.

[55] R. Rieber, M. McHenry, P. Twu, and M. M. Stragier, "Planning for a martian road trip-the Mars2020 mobility systems design," in *Proc. IEEE Aerosp. Conf.*, 2022, pp. 1–18.
[56] D. Bhadauria, O. Tekdas, and V. Isler, "Robotic data mules for collecting data over sparse sensor fields," *J. Field Robot.*, vol. 28, no. 3, pp. 388–404, 2011.
[57] M. Bajracharya, M. W. Maimone, and D. Helmick, "Autonomy for Mars rovers: Past, present, and future," *Computer*, vol. 41, no. 12, pp. 44–50, Dec. 2008.
[58] K. Bestuzheva et al., "The SCIP optimization suite 8.0," 2021, *arXiv:2112.08872*.
[59] R. Mattingly and L. May, "Mars sample return as a campaign," in *Proc. Aerosp. Conf.*, 2011, pp. 1–13.
[60] J. Schoolcraft, A. T. Klesh, and T. Werne, "Marco: Interplanetary mission development on a cubesat scale," in *Proc. 14th Int. Conf. Space Oper.*, 2016, Art. no. 2491.
[61] A. M. Korzun, G. F. Dubos, C. K. Iwata, B. A. Stahl, and J. J. Quicksall, "A concept for the entry, descent, and landing of high-mass payloads at Mars," *Acta Astronautica*, vol. 66, no. 7, pp. 1146–1159, 2010.
[62] V. Kerzhanovich et al., "Breakthrough in Mars balloon technology," *Adv. Space Res.*, vol. 33, no. 10, pp. 1836–1841, 2004.
[63] J. Breidenthal, H. Xie, C.-W. Lau, and B. MacNeal, "Space and earth terminal sizing for future mars missions," in *Proc. SpaceOps Conf.*, 2018, Art. no. 2426.
[64] S. J. I. Herzig, S. Mandutianu, H. Kim, S. Hernandez, and T. Imken, "Model-transformation-based computational design synthesis for mission architecture optimization," in *Proc. IEEE Aerosp. Conf.*, 2017, pp. 1–15.
[65] P. Santana, T. Vaquero, C. Toledo, A. Wang, C. Fang, and B. Williams, "Paris: A polynomial-time, risk-sensitive scheduling algorithm for probabilistic simple temporal networks with uncertainty," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2016, vol. 26, pp. 267–275.

**Joshua Vander Hook** received the Ph.D. degree in computer science from the University of Minnesota, Minneapolis, MN, USA, in 2015.

He was previously a Technical Group Supervisor with the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA. He is currently a Manager of Applied Science with Robotics AI, Amazon.

**Federico Rossi** (Member, IEEE) received the M.Sc. degree in space engineering from Politecnico di Milano, Milan, Italy, in 2013, and the Ph.D. degree in aeronautics and astronautics from Stanford University, Stanford, CA, USA, in 2018.

He is currently a Robotics Technologist with Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA. His research focuses on optimal control and distributed decision-making in multiagent robotic systems, with applications to robotic planetary exploration.

**Tiago Vaquero** received the B.Sc., M.Sc., and Ph.D. degrees from the University of Sao Paulo, Sao Paulo, Brazil, in 2003, 2007, and 2011, respectively, all in mechatronics engineering.

He is currently a Technical Group Leader of the Artificial Intelligence, Observation Planning and Analysis Group, Planning and Execution Section, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA. His research interests include multiagent coordination methods for multirover cave exploration and surface site characterization.

**Martina Troesch** received the B.S. and M.S. degrees in aerospace engineering from the University of Southern California, Los Angeles, CA, USA, in 2011, and the M.S. degree in computer science from Stanford University, Stanford, CA, USA, in 2015.

She is a Software Engineer with Google, Mountain View, CA, USA. She was previously a Member of the Artificial Intelligence Group, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA.

**Marc Sanchez Net** received the degree in telecommunications engineering and industrial engineering from Universitat Politecnica de Catalunya, Barcelona, Spain, in 2012, and the Ph.D. degree in aerospace engineering from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2017.

He is currently a Telecommunications Engineer with the Communication Architectures and Research Section, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA. He studies delay tolerant networking and its impact on distributed applications, such as computational task sharing, spacecraft constellation management, as well as design of space communication systems in challenged environments, such as the surface of the Moon or Mars.

**Joshua Schoolcraft** received the B.S. degree in electrical engineering and computing engineering (with a CS minor) from the University of Maine, Orono, ME, USA, in 2005, and the M.S. degree in aeronautical and astronautical engineering from Stanford University, Stanford, CA, USA, in 2011.

He is an Engineer with the Communications Networks Group, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA. He currently runs the Protocol Technology Laboratory and coordinates software configuration and development on projects involving protocol characterization and DTN.

**Jean-Pierre de la Croix** received the Ph.D. degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2015.

He is currently a Robotics Systems Engineer in the Maritime and Multi-Agent Autonomy Group, Jet Propulsion Laboratory (JPL), California Institute of Technology, Pasadena, CA, USA. He continues to research multiagent systems at JPL.

**Steve Chien** received the B.Sc., M.S., and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign, Champaign, IL, USA.

He is currently a JPL Fellow and a Senior Research Scientist with the Artificial Intelligence Group, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA, where he leads efforts in automated planning and scheduling for space exploration.