JOURNAL OF AEROSPACE INFORMATION SYSTEMS
Vol. 20, No. 8, August 2023

Decentralized Observation Allocation for a Large-Scale Constellation

Shreya Parjan* and Steve A. Chien†
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California 91109-8099

https://doi.org/10.2514/1.I011215

Increased space sensing enables new measurements of a wide range of Earth science phenomena including volcanism, flooding, wildfires, and weather. Large-scale observation constellations of hundreds of assets have already been deployed (for example, Planet Labs's Dove satellites), and several constellations of tens of thousands of assets are planned. New challenges exist to rapidly assimilate available data and to optimize measurements by directing spacecraft assets to best observe complex Earth science phenomena. Centralized approaches to managing request allocation in these large constellations are constrained by 1) the need to assign/elect a central node to assign requests to spacecraft and 2) reliance on a single agent communicating with potentially thousands of dependent agents. On the other hand, entirely decentralized approaches to request allocation and observation are prone to oversatisfaction of some requests and undersatisfaction of others due to a lack of communication among agents. In large constellations, an intermediary method is necessary to solve the request allocation problem in a distributed manner. We present distributed artificial intelligence/multiagent methods that leverage existing work on distributed constraint optimization to allocate observations in a satellite constellation. We compare their performance to centralized and highly decentralized approaches using realistic orbits and observation request distributions. Our distributed algorithms can find approximate solutions to the large-scale constellation request allocation problem with low data volume for agent coordination and extend to continuous planning problems with varying request sets and availability of spacecraft agents.

I. Introduction

WORLDWIDE, there is an explosion of information sources relevant to environmental monitoring. Ground-based sensors are being deployed at an incredible rate, and their data are more easily accessible via the internet of things (IOT). This explosion of networked sensors even extends to space, where traditional and new space actors have deployed worldwide monitoring assets such as Terra; Aqua; Suomi-National Polar-orbiting Partnership (NPP); Sentinel; and Planet Lab's Dove, SkySat, and other constellations (with over 100 spacecraft).

With many potential information sources and space assets come two distinct problems: 1) combining the many information sources to track complex spatiotemporal science phenomena; and 2) tasking the large set of space assets with varying orbits, costs, and capabilities.

Elsewhere, the end-to-end sensor-web concept has been described [1], including deployments to track flooding [2,3], volcanic activity [4], and wildfires [5]. However, those pilots did not study control of constellations because they used only the Earth Observing-1 satellite under direct sensor-web control, although they did submit requests to commercial providers in a federated approach [6]. More recent work in automated scheduling of constellations includes operational control of Dove [7] and Skybox (later SkySat) [8] as well as recent studies on coordination in large-scale constellations like the work described in this paper [9–11]. Most of these describe centralized resource allocation approaches, although notable exceptions exist [11,12].

However, centralized approaches to constellation operations are vulnerable to loss of the central node and unreliable communication to/among spacecraft. Some centralized approaches include leadership election, in which issues with the central leader are detected and result in the election of a new leader. Nonetheless, detecting issues with a centralized leader can be challenging and affect nominal performance. Election of a new leader/central node can be a time-consuming process. Distributed resource allocation avoids these issues, albeit often at a decrease in performance.

Motivated by work on the maximum gain messaging (MGM) algorithm and the distributed stochastic algorithm (DSA), which are two incomplete distributed constraint optimization problem (DCOP) algorithms [13,14], we present two types of broadcast decentralized (BD) algorithms for a less computationally intensive, heuristic search-based approach to observation request allocation in large-scale satellite constellations: BD request satisfaction, and BD contention. We aim to maximize the number of requests satisfied by agent observations and minimize the number of future requests agents cannot observe due to data volume or slew constraints.

In this paper, Sec. II describes the request allocation problem for large-scale constellations, Sec. III offers an overview of various approaches to solving the problem and introduces our broadcast decentralized algorithms, Sec. IV explores the application of these algorithms to a more realistic continuous planning problem, Sec. V describes results from experiments evaluating the algorithms on an allocation problem involving thousands of observation requests distributed among hundreds of satellites, and Sec. VI proposes avenues for future work.

II. Problem Formulation

A. Spacecraft Operations Scheduling

We study the problem of allocating observation requests to satellites.‡ We define the observation scheduling problem as follows:

The problem inputs are listed as follows:

1) $[H_s, H_e]$ is the scheduling horizon, starting at time H_s and ending at time H_e.

2) A is a set of agents $\{a_1 \ldots a_m\}$, where individual spacecraft are the primary agents in our implementation. (In our current formulation, an agent represents a single spacecraft; but in alternative formulations (i.e., a federated system), an agent could represent multiple spacecraft or there could be multiple levels of hierarchy.

Received 2 November 2022; revision received 27 February 2023; accepted for publication 6 March 2023; published online 10 April 2023.

*Member of Technical Staff, Artificial Intelligence Group, 4800 Oak Grove Drive: shreya.parjan@jpl.nasa.gov (Corresponding Author).

†Jet Propulsion Laboratory Fellow, Senior Research Scientist, and Technical Group Supervisor, Artificial Intelligence Group, 4800 Oak Grove Drive.

‡Although we target the allocation of observations to satellites in an Earth observation satellite constellation, our approach does generalize to other problems.

Fig. 1 Global distribution of point targets.

Table 1 Broadcast decentralized algorithmic variations

Feature	Broadcast decentralized contention	Broadcast decentralized request satisfaction
Required shared information for iteration-phase update procedure	Reward for observing request (float) Count of remaining free overflights for request (integer) Request satisfaction (Boolean)	Request satisfaction (Boolean)
Allocation initialization options $P_{initialize}$	P_{ratio} = (no. of agent overflights for request)/(no. of constellationwide overflights for request) $P_{totalOFs}$ = 1/(no. of constellationwide overflights for request) P_{random} = user-specified probability	P_{random} = user-specified probability
Local request sort variations	Iterative global free overflight (GFO) sort Iterative local free overflight sort Iterative random sort $W_{difference}$ = (no. of agent overflights for request)-(no. of agent conflicts for overflight selected for request)	Iterative local free overflight (LFO) sort Iterative random (RD) sort
Reward function variations for iteration-phase update procedure	W_{ratio} = (no. of agent overflights for request)/[1+ (no. of agent conflicts for overflight selected for request)]	Not Applicable (reward not used)

Furthermore, for complex spacecraft platforms with many instruments, a spacecraft might be represented by multiple agents.)

3) K is a set of orbits $\{k_1 \ldots k_m\}$: one for each spacecraft agent in A.

4) T is a set of point targets $\{t_1 \ldots t_o\}$, with each defined by a name and a single pair of coordinates (Fig. 1).

5) R is a set of requests $\{r_1 \ldots r_p\}$, where a request r_c is defined by which target t_b to observe and when in the scheduling horizon $[H_s, H_e]$ the user would like t_b to be observed.

From the preceding, we generate a set of overflights where each overflight is an opportunity for a spacecraft to view a specific target to satisfy a request. Each such overflight has both an associated status indicating if (in the current schedule) the spacecraft elects to satisfy the request at that opportunity and an associated reward for that satisfied overflight–request pair.

6) O is a set of overflights $\{O_{111} \ldots O_{hij}\}$. O_{hij} specifies a time[¶] at which agent a_h could potentially schedule request r_i. To distinguish between multiple overflights by the same agent for the same request, we use an additional overflight index j. Each agent's overflights must be during the daytime (i.e., have an associated solar zenith angle of greater than 90).

7) S is a set of statuses $\{s_{11} \ldots s_{mp}\}$. Note that s_{mp} has a value of one if agent a_m schedules an observation to satisfy request r_p and a value of zero otherwise.

8) W is a set of computed rewards $\{w_{11} \ldots w_{mp}\}$. Note that w_{mp} specifies the reward that agent a_m reaps for satisfying request r_p. We expand on the reward functions in Table 1.

B. Scheduler Objective

A problem solution is an assignment of observations to satellites (agents) and, for each satellite, an assignment of observations to satisfying overflights. Complicating request satisfaction are the following operations constraints enforced as described in the following:

1) The first constraint is *slew*. In our experiments, we use a simple model requiring $|O_{amj} - O_{ank}| > offset$, where O_{amj} and O_{ank} are overflights by the same spacecraft for different requests and offset may be specified by the user. We use offset=30 s. [However, our approach extends to a higher-fidelity model in which this constraint would depend upon the pointings required to image the consecutive targets and spacecraft agility. Specifically, an arbitrary constraint enforcement function would compute mutually exclusive overflights to be enforced by the relevant agent(s) during scheduling.]

2) The second constraint is *data volume*. For a simple implementation of spacecraft data volume constraints, we enforce that a spacecraft cannot schedule more than n_{cap} requests during the scheduling horizon. In our experiments, $n_{cap} = 1.5 \times |R|/|A|$; so, no single spacecraft may schedule observations to satisfy more than 150% of the requests allocated in an evenly divided share of requests.[§]

Given the preceding formulation, the scheduler's objective is to select a subset of the total overflights to maximize some function of satisfied requests. In this preliminary formulation, the objective is maximization of the number of requests satisfied; but, more realistic formulations incorporating request priority, fairness, and other metrics are clear areas for future work. A solution has the following form:

[¶]Indeed, more expressive observation campaigns are envisioned as future work (e.g. as in [15] and [16]).

[§]Again, more complex data volume constraints and explicit handling of downlink activities could be incorporated as future work.

$$\max \sum_{i=1}^{m} \sum_{j=1}^{p} w_{ij} \tag{1}$$

such that Eqs. (2) and (3) hold:

$$s_{id} + s_{ie} \leq 1 \text{ if} |O_{idj} - O_{iek}| \leq offset \tag{2}$$

where O_{idj} and O_{iek} are overflights selected by agent a_i to observe each pair of requests, r_d and r_e, and

$$\sum_{j=1}^{p} s_{ij} \leq n_{\text{cap}} \tag{3}$$

for each agent a_i.

III. Range of Approaches

We present and analyze several algorithms for the observation request–overflight allocation problem. For a given target, a request represents a single desired observation (e.g., "observe on day 3"). A set of requests may, together, comprise a desired repeat observation (e.g., "observe every hour") during the scheduling horizon $[H_s, H_e]$. [Indeed, more expressive observation campaigns are envisioned as future work (e.g., as in Refs. [15] and [16]).] We define three algorithm families: centralized, broadcast decentralized, and highly decentralized. We further subdivide broadcast decentralized into a contention-based algorithm and a request satisfaction-based algorithm, which are differentiated by the amount of state information shared among agents (Fig. 2). Our focus is on evaluating algorithms from the broadcast decentralized family, which is derived from distributed constraint optimization algorithms, and comparing them to one another as well as to the centralized and highly decentralized algorithms at either extreme of the centralization and information sharing spectra.

A. Limitations of Centralized and Highly Decentralized Approaches

Extensive existing work has solved the problem of observation request allocation in a satellite constellation using centralized approaches [17]. Such solutions rely on a scheduler tasking a constellation of satellites. The request allocation problem is solved at this central scheduling node that has access to satellite overflight and request information. The satellite agents receive and execute their final, computed schedules from the scheduling node. If the scheduling agent is itself a satellite, as is the case in a hierarchical or leadership election implementation (where a subset of satellites conducts scheduling on behalf of the entire constellation), the network is susceptible to faults if the central node is lost, much like if all request allocations are done by a single ground station that faces the additional cost of uplinking each agent's computed schedules. Although it may be possible to elect another leader from the remaining network, the election process may be costly in time and communications resources, which could penalize constellation performance and result in missed observation opportunities.

For comparison, we also implement a "highly decentralized" algorithm in which satellite agents each receive the initial set of observation requests from a central source (this sharing distinguishes highly decentralized from "[completely] decentralized") and conduct scheduling in isolation, without exchanging any state information. Without any coordination among agents, some requests may be oversatisfied, with multiple agents selecting overflights to schedule them, whereas other requests may be undersatisfied if agents reach their data volume capacity before attempting to schedule them. Our aim in implementing an approach with such clear limitations is to contrast with our centralized algorithm and show how our broadcast decentralized algorithms can provide a necessary middle ground.

Finally, we introduce our suite of broadcast decentralized algorithms in an attempt to reap the benefits of distributed scheduling but use limited interagent communication to retain some algorithmic efficiency. Namely, our decentralized approach can handle cases where an agent drops out of or is added to the constellation without significant disruption to existing scheduling. Furthermore, by exchanging information to decide whether to bid on a request, agents avoid some of the inefficiencies of the highly decentralized approach. Note that in the broadcast decentralized algorithms, data consistency must be maintained for agents to make decisions on which requests to bid on as compared to their peers. We address this challenge by having agents update which requests they bid on based on stored, shared state information from the previous iteration of scheduling. We summarize each of the aforementioned algorithm families in Table 2 and discuss our implementations of each in the subsequent sections.

In Secs. III.B, III.C, and III.E, we reference the following functions:

1) The first function is $ASSIGN(r_i, a_j)$: If agent a_j is not at capacity, instruct a_j to select its overflight for request r_i that conflicts with the fewest overflights a_j has for other requests. If successful, the agent will mark r_i as satisfied, rule out its overflights that conflict with the identified overflight, add the identified overflight to its schedule of observations, and increment the count of requests it is assigned to observe in the scheduling horizon. This calls $ELIMINATE\_CONFLICTS(assignment_{a_j,r_i})$ and returns the agent's observation schedule.

2) The second function is $SELF\_ASSIGN(r_i, P_{initialize})$: The current agent checks its set of daytime overflights (with a solar zenith angle of greater than 90 deg) to see if it has overflights for request r_i. If so, the agent changes its self-assignment status for request r_i from zero (false) to one (true) with probability $P_{initialize}$ (Table 1). This returns a status of zero or one.

3) The third function is $BROADCAST(r_i, a_j, items)$: The current agent a_j instructs all agents to record its values for the state variables in $items$ for request r_i in the current iteration of the algorithm. In BD request satisfaction, $items$ contains only a Boolean for whether a_j

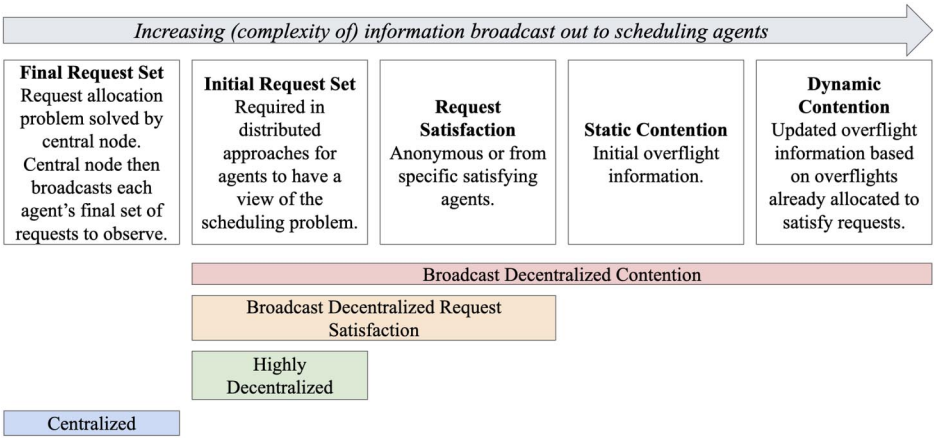

Fig. 2 Interagent information sharing by algorithm.

Table 2 Comparison of various request allocation approaches

	Centralized/hierarchical	Leadership election/ hierarchical	Broadcast decentralized	Highly decentralized
Solver for request allocation layer	Single, centralized scheduling node.	Multiple nodes/satellite agents handle scheduling on behalf of constellation.	Individual satellite agents bid on requests.	Individual satellite agents "pick up" requests they have nonzero overflights for.
Agent overflight allocation layer	Scheduling node selects agent overflights.	Superordinate "leader" agents select overflights for subordinate agents.	Overflights allocated to requests by individual agents using a common algorithm.	Individual agents attempt to schedule every request they picked up while maintaining data volume constraint.
Shared information	Satellite agents receive their final execution schedule from a single scheduling node.	Satellite agents receive their final execution schedules from scheduling node(s).	Initial request set, agent request satisfaction, agent overflights, contention metrics. $O(A^2)$ messages exchanged.	Initial request set.
Vulnerabilities	Subcase of hierarchical: threat of master node loss.	Augmentation of centralized/ hierarchical algorithm; leadership reelection is costly.	Data consistency must be maintained.	Very inefficient allocations. Risk of request oversatisfaction and wasted agent capacity.

found an overflight to satisfy r_i. In BD contention, *items* includes that Boolean in addition to the reward a_j reaped for satisfying r_i and the agent's number of remaining, free overflights for the request (Table 1).

4) The fourth function is $SCHEDULE(r_i)$: The current agent searches its set of available overflights for request r_i for one with a time stamp that conflicts with the fewest overflights it has for other requests, per the slew constraint. If there are multiple such overflights, the agent selects the one with the earlier time stamp. If the search is successful, the agent marks r_i as satisfied, adds the identified overflight to its schedule of observations, and increments the number of requests it is assigned to observe in the scheduling horizon (to later check against the data volume constraint). This returns 1) the assignment as a tuple with form (spacecraft identifier, request index, or overflight time) and 2) the number of conflicts the selected overflight has with the current agent's other overflights.

5) The fifth function is $ELIMINATE\_CONFLICTS(assignment_{a_j,r_i})$: Given the overflight time selected by agent a_j to observe request r_i in the tuple $assignment_{a_j,r_i}$, agent a_j records which of its overflights are no longer feasible if request r_i is scheduled because selecting them would violate the slew constraint.

6) The sixth function is $UPDATE\_STATUS(r_i, P_{assign}, P_{unassign})$: Based on whether the current agent satisfied request r_i in the previous iteration of scheduling, it either unassigns itself from request r_i with probability $P_{unassign}$ or assigns itself to request r_i with probability P_{assign}.

7) The seventh function is $SHUFFLE(R)$: This randomizes the order of the list of requests R if the local request sort procedure is an iterative random sort (Table 1).

8) The eighth function is $SORT(R)$: This sorts the list of requests R using one of the sorting methods named in Table 1.

B. Algorithm 1: Centralized

The application of centralized, hierarchical, or auction-based solvers managed by a central scheduling node is well documented in existing work on solving the observation request allocation problem [9,10,12]. Thus, to demonstrate the viability of our broadcast decentralized algorithms, it is important to compare their performances on large-scale scheduling problems to that of their centralized counterparts.

Our baseline centralized algorithm (Fig. 3), Algorithm 1, solves the request allocation problem at a master node. That is, one central node (i.e., another satellite or a ground station) solves the scheduling problem for all agents. Algorithm 1 considers requests in ascending order by an assigned ranking. The initial ranking for a request is determined by the total number of overflights that all agents have for it in the scheduling horizon. From R, the scheduling node creates a min-heap (minimum binary heap)$R_{minHeap}$ such that the top of $R_{minHeap}$ is initially the request with the fewest overflights in the scheduling horizon. Each request r_i in R also maintains a max-heap (maximum binary heap) $A_{maxHeap,r_i}$ of agents with overflights for r_i. The agent at the top of $A_{maxHeap,r_i}$ has the most overflights for r_i.

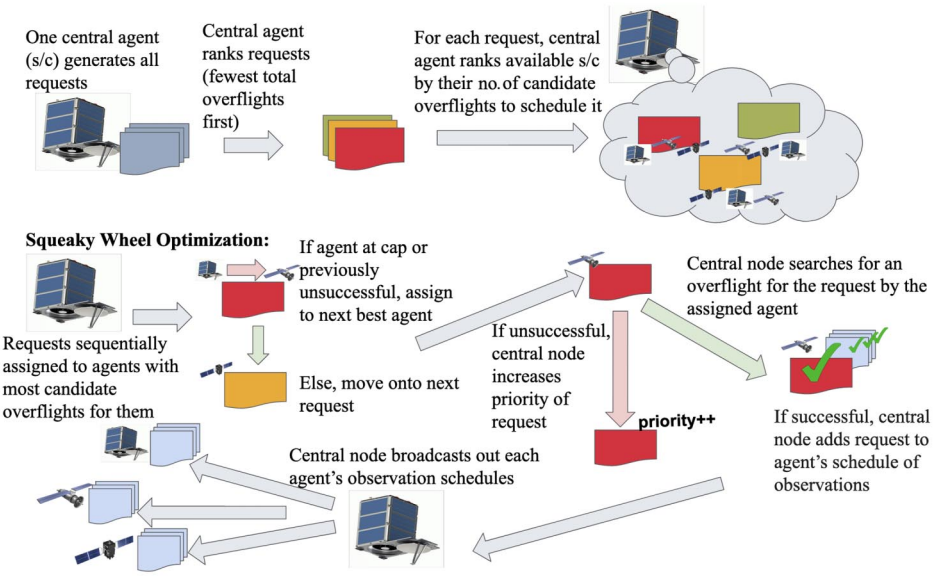

Fig. 3 Pictorial representation of our centralized algorithm with squeaky wheel optimization (s/c = spacecraft).

Algorithm 1: Centralized

Input: A set of requests R, a set of agents A, and a set of overflights O.
Output: L, which is a set of schedules for each agent $a_j \in A$.

while $iteration < maxIterations$, do
  create $R_{minHeap}$
  while $|R_{minHeap}| > 0$, **do**
    pop r_0 **from** $R_{minHeap}$
    if $totalOverflights_{r_0} > 0$, **then**
      create $A_{maxHeap,r_0}$
      while $|A_{maxHeap,r_0}| > 0$ **and** r_0 *unsatisfied*, **do**
        pop a_{max,r_0} **from** $A_{maxHeap,r_0}$
        $L \leftarrow ASSIGN(r_0, a_{max,r_0})$
      end
    end
  end
  $iteration \leftarrow iteration + 1$
end
return L

Algorithm 2: Highly decentralized

Input: A set of requests R, a set of agents A, and a set of overflights O.
Output: L, which is a set of schedules for each agent $a_j \in A$.

while $iteration < maxIterations$, **do**
  for each agent $a_j \in A$, **do**
    $SHUFFLE(R)$
    for $r_i \in R$, **do**
      if a_j *not at capacity*, **then**
        $assignment_{a_j,r_i}, numConflicts_{a_j,r_i} = SCHEDULE(r_i)$
        if $assignment_{a_j,r_i}$ *valid*, **then**
          $L \leftarrow assignment_{a_j,r_i}$
          $ELIMINATE\_CONFLICTS(assignment_{a_j,r_i})$
        end
      end
    end
  end
  $iteration \leftarrow iteration + 1$
end
return L

Once the central node has constructed a solution to the scheduling problem, it sends out the resulting observation schedules to each agent.

For the scheduling node to construct the solution, we leverage squeaky wheel optimization (SWO) [18] in our centralized algorithm. SWO allows us to greedily construct an initial solution that can be improved upon iteratively by increasing the priority of unsatisfied requests. This mirrors the logic by which a ground station might reprioritize requests based on downlinked information describing which observations were successfully captured.

The central node pops the request with the lowest ranking r_0 from $R_{minHeap}$ and assigns it to the agent with the most free overflights for it in the scheduling horizon. If that agent is not at capacity, the central node considers the request with the next-lowest ranking. Otherwise, it attempts to assign r_0 to the agent with the next-most overflights for it in the scheduling horizon. It continues to reattempt this before moving onto the next request until either r_0 is satisfied or there are no remaining agents with overflights for r_0. Because the central node manages state information for all agents over iterations of scheduling, a limitation of this implementation is that we are not able to parallelize the algorithm. If lost or compromised, the central node can be reelected, though this process may be time consuming.

Once all requests have been considered, the central node increases the ranking assigned to a request based on whether it was satisfied or not. If the request was satisfied, its ranking is increased. If the request was not satisfied, its ranking is assigned to zero, to move it to the top of $R_{minheap}$ in the following iteration of the algorithm, after $R_{minHeap}$ is created once more (thereby "greasing the squeaky wheel"). The algorithm iterates until a user-specified number of iterations are reached. Users may also specify an early exit condition, such as when all satellites are subscribed at capacity, all observation requests are satisfied, or the number/set of observation requests satisfied between the current and previous iterations is the same.

C. Algorithm 2: Highly Decentralized

No feedback on request satisfaction or state information is shared among agents in our highly decentralized algorithm (Fig. 4). Thus, the algorithm provides a lower bound for evaluating the distributed broadcast decentralized approaches at the heart of this paper. For this single-pass algorithm, each request is broadcast out to all agents. From there, agents independently schedule observations of requests based on their local view of the problem alone. That is, an agent will allocate itself to every request it has nonzero overflights for and proceed to search for the overflight satisfying the current request that conflicts with the fewest other overflights it has for other requests. Because selecting an overflight to observe a request may eliminate other conflicting overflights (to maintain the slew constraint), the order in which requests are considered matters. Thus, each agent randomizes the order in which it considers and attempts to schedule requests. In our current implementation of Algorithm 2, requests do not have associated priorities. If an overflight is successfully found, conflicting overflights for other requests are eliminated from the agent's set of available overflights.

D. Distributed Constraint Optimization

Our broadcast decentralized algorithms are adapted from the maximum gain messaging algorithm and distributed stochastic algorithm for solving distributed constraint optimization problems [19].

DCOPs are defined as tuples $P = (A, X, D, C, \alpha)$:

1) $A = \{a_1 \ldots a_p\}$ denotes a set of agents.
2) $X = \{x_1 \ldots x_n\}$ denotes a set of variables such that $n \geq m$.
3) $D = \{D_1 \ldots D_n\}$ denotes a set of domains for the corresponding $x_i \in X$.

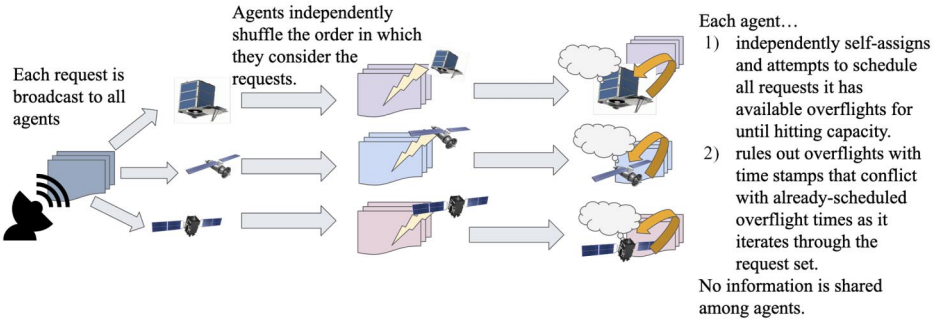

Fig. 4 Pictorial representation of the highly decentralized algorithm.

4) $C = \{c_1 \ldots c_k\}$ denotes a finite set of constraint functions involving the variables in X.

5) For tuple α: $X \rightarrow A$, which is a mapping of variables to agents.

Complete solutions to DCOPs assign a value to each $x_i \in X$ while satisfying all constraint functions in C.

For our problem, we have the following:

1) $A = \{a_1 \ldots a_m\}$ denotes the set of spacecraft agents.

2) $X = \{o_{11} \ldots o_{mp}, s_{11} \ldots s_{mp}, w_{11} \ldots w_{mp}\}$ includes variables where, for example, agent a_m has o_{mp} free overflights for request r_p and, if $s_{mp} = 1$, a_m schedules request r_p with reward w_{mp}.

3) $D = \{D_1 \ldots D_{3\times m\times p}\}$ denotes a set of domains for the variables in X: the free overflight counts and rewards take on nonnegative integer values, and the scheduling variables are binary decision variables with domain $\{0, 1\}$.

4) $C = \{c_1, c_2\}$, where for some agent a_i, a) c_1: $s_{id} + s_{ie} \leq 1$ if $|O_{idj} - O_{iek}| \leq offset$ (where O_{idj} and O_{iek} are the overflights selected by agent a_i to observe each pair of requests r_d and r_e); and b) c_2: $s_{i1} + \ldots + s_{ip} \leq n_{cap}$.

As before, our objective [Eq. (1)] is to maximize the reward across all agents for satisfying each request in R while adhering to the constraints in C.

Both the MGM algorithm and the DSA are incomplete, synchronous search-based algorithms [13,14]. In the MGM algorithm, an agent communicates exclusively with its neighbors: agents whose variables appear in the same cost function(s) as that agent's own variables. Under our problem formulation, for each request and at each iteration, the set of neighbors would be all agents with free overflights for the request. Each agent initializes its variable values randomly and repeats the following procedure until some termination condition is reached:

1) Broadcast variable values out to all neighbors.

2) Receive values of neighbors' variables and compute the maximum gain obtained by changing own variable values.

3) Broadcast that gain out to all neighbors.

4) Receive neighbors' gains and update value if own gain is the largest.

The DSA is similar, but instead of steps 3 and 4, the agents stochastically decide whether to take on the variable values associated with maximum gain.

Combining the stochastic updating procedure from the DSA and the more informed updating procedure of the MGM algorithm, the agents in the broadcast decentralized algorithms update their self-assignment statuses for a particular request in the semistochastic manner outlined in Figs. 6 and 7 for the contention and request satisfaction algorithms, respectively. This helps avoid local maxima when optimizing for the number of successfully scheduled requests over iterations of the algorithms.

The generic DCOP algorithms and our BD algorithms differ in the order in which agents receive broadcasts. To maintain a consistent view of the problem across all agents for each iteration, our agents broadcast state information out to all others. Broadcasting information to all agents increases the overall number of messages exchanged but eliminates the need to continuously recompute or reverify agents' sets of neighbors. An excellent optic for future work is the investigation of methods for computing neighborhoods to limit interagent communication to an "as-needed" basis (in contrast to our "broadcast to all" approach).

E. Algorithm 3: Broadcast Decentralized

Existing work [11,12] describes parallel single-item auctions and sequential single-item auctions that rely on a single auctioneer agent. In the broadcast decentralized algorithms, like the consensus-based bundle algorithm (CBBA) [12], coordination instead occurs exclusively between scheduling agents that may vary in their orbits, costs, and capabilities (Fig. 5). Unlike the CBBA approach, our BD algorithms consider requests iteratively, eliminating the overhead of constructing and evaluating bundles of requests. Another existing approach applies the distributed pseudotree optimization procedure (a complete algorithm) strictly to solving the observation request allocation problem for superuser agents with exclusive control of certain satellite orbit portions [11]. In contrast, our MGM- and DSA-inspired methods use a heuristic, semistochastic approach for satellite agents to update whether they are self-assigned to attempt to schedule an overflight for a request. Agents make this decision based on shared information about request satisfaction alone (BD request satisfaction) or request satisfaction along with the reward and number of excess overflights for the current request (BD contention). Such an approach trades solution quality for the ability to scale up to larger distributed constraint optimization problems involving hundreds of agents.

In the iterative centralized algorithm, the central node shares only each agent's schedule of requests to observe after each invocation of the scheduler. In the broadcast decentralized algorithms, the initial set of requests must be broadcast to each agent so it can establish a local view of the scheduling problem. The highly decentralized algorithm is a single-pass algorithm in which all agents receive the set of requests and attempt to schedule them without any interagent communication.

Our broadcast decentralized algorithms consist of two layers: allocation and search. In allocation, agents determine how many overflights they have during the scheduling horizon for each request r_i. In the BD contention algorithm, this is broadcast out for all other agents to record for the initial iteration of the algorithm. Agents with

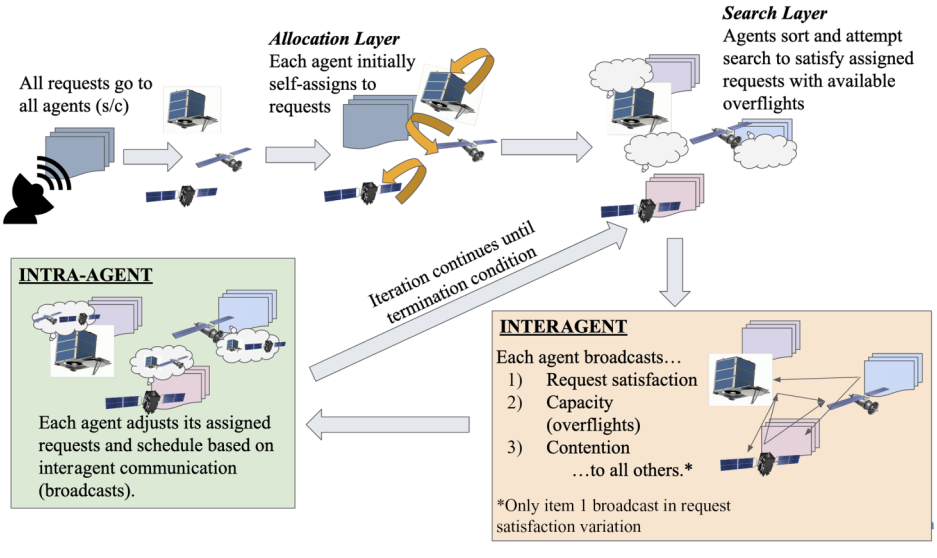

Fig. 5 Pictorial representation of the generic broadcast decentralized algorithm.

overflights for r_i self-assign to find an overflight to observe it with some probability $P_{initialize}$ (Table 1).

In the search phase, if a request was satisfied by multiple agents' overflights in the previous iteration, excess agents stochastically unassign from it. If it was unsatisfied, additional unassigned agents probabilistically self-assign to attempt to schedule the request in the current iteration. Agents update whether they are self-assigned to observe a request in the current iteration based on state information from the previous iteration of the algorithm. This allows agents to avoid race conditions in both our current single-threaded simulation of parallel execution and in a future parallelized implementation. In BD request satisfaction, agents broadcast whether they satisfied a request and update whether they are self-assigned to attempt to schedule it in the current iteration based on how many other agents satisfied it. In BD contention, agents broadcast whether they satisfied a request, along with the number of free overflights they have remaining for that request and their reward for satisfying it (a function of the number of free overflights and the number of conflicts the selected overflight has with the agent's overflights for other requests).

We introduce two novel broadcast decentralized algorithms with 10 potential variations along the following axes: required shared information, agent allocation initialization, agent request sort procedure, and agent reward for request satisfaction (Table 1). First, our approaches vary in the amount of information shared between agents. The BD contention algorithm requires agents to share whether they satisfied a request, their reward for doing so, and the number of free overflights they have remaining for that request every iteration (Algorithm 4). BD request satisfaction only requires agents to broadcast whether or not they satisfied a request (Algorithm 3).

In both broadcast decentralized algorithms, agents iterate through the set of requests broadcast to them by the central node. Because a selected overflight to schedule one request may eliminate conflicting overflights that an agent has for another request, the order in which requests are considered matters. For each of the two broadcast decentralized algorithms, the variations differ in their sort functions

Algorithm 3: Broadcast decentralized request satisfaction

Input: A set of requests R, a set of agents A, and a set of overflights O.
Output: L, which is a set of schedules for each agent $a_j \in A$.

for each agent $a_j \in A$, **do**
  $SORT(R)$
  for *each request* $r_i \in R$ *that* a_j *has overflights for*, **do**
    $assignmentStatus_{a_j,r_i} = SELF\_ASSIGN(r_i, a_j, P_{initialize})$
  end
end
while $iteration < maxIterations$, **do**
  for *each agent* $a_j \in A$, **do**
    for *each request* $r_i \in R$, **do**
      if *not on the first iteration*, **then**
        $UPDATE\_STATUS(r_i, P_{assign}, P_{unassign})$
      end
      if $assignmentStatus_{a_j,r_i} == 1$, **then**
        $assignment_{a_j,r_i}, numConflicts_{a_j,r_i} = SCHEDULE(r_i, a_j)$
        if $assignment_{a_j,r_i}$ valid, **then**
          $L \leftarrow assignment_{a_j,r_i}$
          $ELIMINATE\_CONFLICTS(assignment_{a_j,r_i})$
          $ofFound_{a_j,r_i} = True$
          $BROADCAST(r_i, a_j, [ofFound_{a_j,r_i}])$
        end
      end
    end
  end
  $iteration \leftarrow iteration + 1$
end
return L

Algorithm 4: Broadcast decentralized contention

Input: A set of requests R, a set of agents A, and a set of overflights O.
Output: L, which is a set of schedules for each agent $a_j \in A$.

for *each agent* $a_j \in A$, **do**
  for *each request* $r_i \in R$ *that* a_j *has overflights for*, **do**
    $numOfs_{a_j,r_i} = |[o_{ji1} \ldots o_{jin}]|$
    $BROADCAST(r_i, a_j, [numOfs_{a_j,r_i}])$
  end
end
for *each agent* $a_j \in A$, **do**
  $SORT(R)$
  for *each request* $r_i \in R$ *that* a_j *has overflights for*, **do**
    $assignmentStatus_{a_j,r_i} = SELF\_ASSIGN(r_i, a_j, P_{initialize})$
  end
end
while $iteration < maxIterations$, **do**
  for *each agent* $a_j \in A$, **do**
    for *each request* $r_i \in R$, **do**
      if $iteration > 0$, **then**
        $UPDATE\_STATUS(r_i, P_{assign}, P_{unassign})$
      end
      if $assignmentStatus_{a_j,r_i} == 1$, **then**
        $assignment_{a_j,r_i}, numConflicts_{a_j,r_i} = SCHEDULE(r_i, a_j)$
        if $assignment_{a_j,r_i}$ *valid*, **then**
          $L \leftarrow assignment_{a_j,r_i}$
          $ELIMINATE\_CONFLICTS(assignment_{a_j,r_i})$
          $ofFound_{a_j,r_i} = True$
          $BROADCAST(r_i, a_j, [ofFound_{a_j,r_i}, numConflicts_{a_j,r_i}, reward_{a_j,r_i}])$
        end
      end
    end
  end
  $iteration \leftarrow iteration + 1$
end
return L

but require agents to share the same information. The three sort variations to determine the order in which agents consider requests are as follows:

1) For the iterative global free overflight sort, agents sort requests at the start of each iteration in ascending order by the cumulative number of available overflights all agents have for each request.

2) For theiterative local free overflight sort, agents sort requests at the start of each iteration in ascending order by the number of available overflights they alone have for each request.

3) for the iterative random sort, agents randomize the order of the requests at the start of each iteration.

Because the iterative global overflight sort requires contention information (the number of free overflights for a request) to be shared among agents, the only two broadcast decentralized request satisfaction sort variations are the iterative local free overflight sort and the iterative random sort.

During the initialization phase of BD contention $P_{initialize}$, the probability with which an agent initially self-assigns to attempt to schedule a request can take on one of three forms: P_{random} is a user-specified value between zero and one. P_{ratio} is the ratio of the number of initial overflights the agent has for the request to the total number of initial overflights all agents have for the request. Finally, P_{total_ofs} is a ratio of one to the total number of initial overflights all agents have for the request. Because overflight information is not shared in BD request satisfaction, $P_{initialize}$ must be supplied by the user as P_{random}.

During the iteration phase, the procedures by which agents update whether they are self-assigned to a request for BD contention and BD request satisfaction are outlined in Figs. 6 and 7,

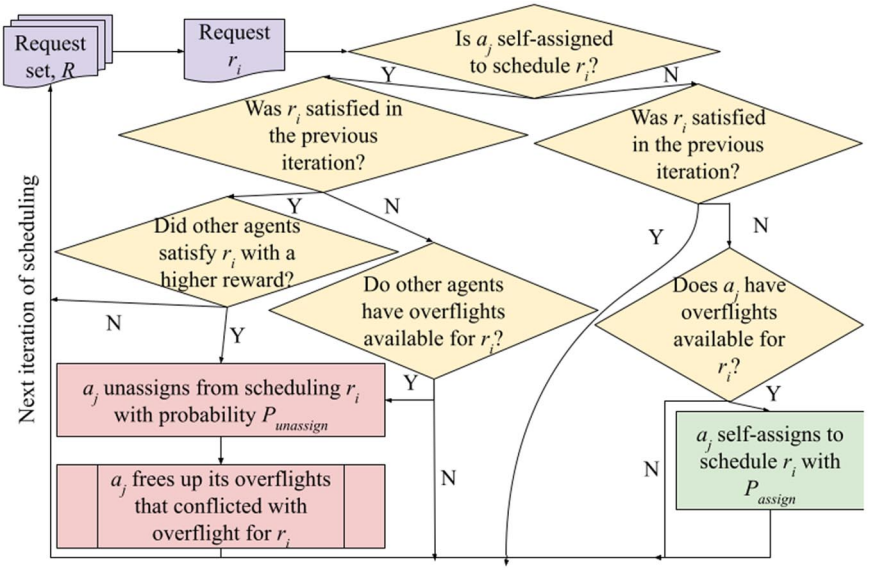

Fig. 6 ***UPDATE_STATUS*(r_i,P_{assign},$P_{unassign}$): agent self-assignment update procedure for BD contention.**

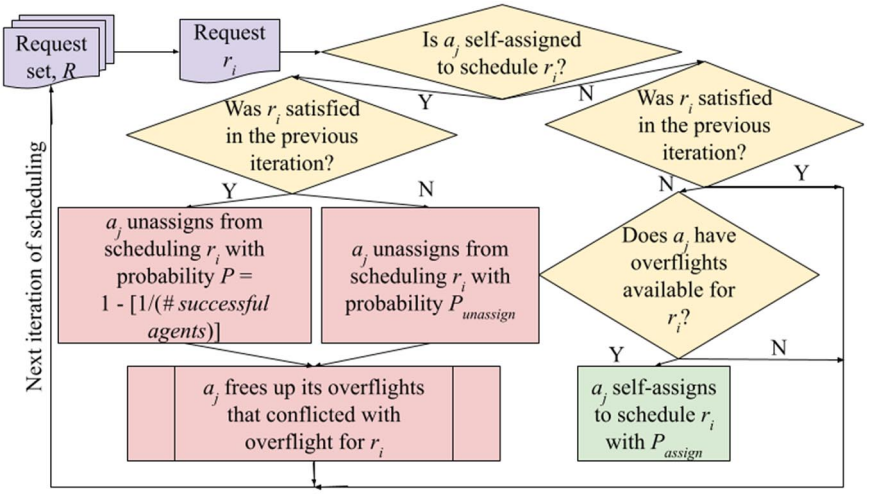

Fig. 7 ***UPDATE_STATUS*(r_i,P_{assign},$P_{unassign}$): agent self-assignment update procedure for BD request satisfaction.**

respectively. In general, agents update whether they are self-assigned to a request in the current iteration based on broadcast information from the previous iteration. BD contention requires agents to consider their reward for satisfying a request in their update procedure. The reward may either be a simple difference between the number of free overflights the agent has for the request and the number of conflicts its selected overflight has with the agent's other overflights or a ratio of the number of free overflights to 1+ the number of conflicts.

The first reward function $W_{difference}$ rewards agents that select an overflight with fewer conflicts with their overflights for other requests to schedule the request and have more free overflights (more excess capacity) for the request. Agents with more capacity are also more likely to have more conflicts, and $W_{difference}$ can take on negative values when the number of conflicts exceeds the number of free overflights. The second reward function W_{ratio} offers users a nonnegative alternative by computing a ratio instead.

When an agent unassigns itself from a request r_u, it must add back the overflights for other requests that conflicted with the overflight it may have selected to observe r_u. In BD contention, the agent must also broadcast the updated number of overflights for each of these other requests to maintain consistency across all agents' views of the problem.

IV. Application to Continuous Planning

The extension of our study to continuous planning offers more realistic operational context for scheduling. In this context, the constellation periodically receives new changes in requests, where such a change might be the addition of a new request or revocation of an existing request. The key insight here is that the set of new requests and requests being revoked is small as compared to the set of requests still valid. Additionally, existing agents may drop from the constellation (e.g., no longer be able to satisfy request) or new agents may be added to the constellation and be available to satisfy requests.

Earlier, we advocated that distributed approaches to observation request allocation in large-scale constellations ease the network's ability to continue scheduling with little disruption in the face of changes in active agents or to the set of requests as scheduling progresses in time. Our application of the broadcast decentralized algorithms to continuous planning aims to highlight this capability as compared to continuous planning conducted with our centralized and highly decentralized algorithms. Furthermore, we specifically introduce two approaches to implementing continuous planning: 1) from scratch continuous planning (FSCP), in which agents schedule all requests from scratch every invocation; and 2) incremental continuous planning (ICP), where agents receive notice of which requests

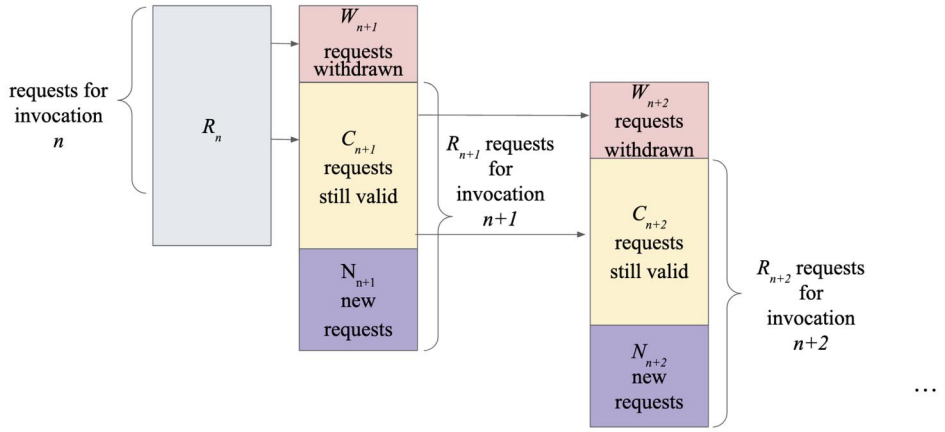

Fig. 8 Pictorial representation of revisions to request set in continuous planning scenarios.

have been added or removed from the request set before the start of another invocation of scheduling and then modify their existing schedules to remove undesired observations and accommodate new ones. We demonstrate that ICP provides better solution quality as compared to FSCP and is more resilient to changes and disruptions in the constellation and scheduling problem.

In our experiments, each invocation of the scheduler corresponds to a one-day step forward in time (Fig. 8). To assess how resilient each continuous planning approach is to changes in desired target observations, odd invocations call for requests from 10% of the targets to be withdrawn. In nonzero even invocations, these targets, along with their corresponding observation requests, are reinstated while another distinct 10% of the targets are withdrawn. This additional set of targets is reinstated in the next odd invocation. Note that the number of observation requests remains unchanged over invocations due to the scheduling horizon sliding one day forward at a time. We similarly alternate which 10% of the spacecraft agents are deactivated and reintroduced into the constellation. In each of our algorithms, deactivated agents cease to schedule any new requests in subsequent invocations. The rest of the network interprets this behavior as the deactivated agents failing to satisfy new requests they are assigned.

The algorithms reference the following functions:

1) The $UPDATE\_REQUESTS(activeRequests, withdrawn\ Requests, H_s, H_e)$ function adds new observation requests to existing *activeRequests* and requests for which the observation is no longer desired or in the window of the current scheduling horizon to *withdrawnRequests*. It returns updated *activeRequests* and *withdrawnRequests*.

2) The $UPDATE\_AGENTS(activeAgents, inactiveAgents)$ function is used to simulate agents joining or dropping out of the satellite constellation. It adds newly accessible agents to *activeAgents* and inaccessible agents to *inactiveAgents*. It returns updated *activeAgents* and *inactiveAgents*.

3) The $SCHEDULE\_ACTIVE\_REQUESTS(activeRequests, activeAgents, O)$ function calls to the desired algorithm (centralized, highly decentralized, or broadcast decentralized) to solve the scheduling problem. It returns a set of schedules for each agent.

4) The $SCHEDULE\_NEWLY\_ACTIVE\_REQUESTS(active\ Requests, activeAgents, O)$ function calls to the desired algorithm (centralized, highly decentralized, or broadcast decentralized) to update the existing solution to the scheduling problem by attempting to schedule only the requests that have newly been set to active (i.e., either did not exist or were inactive in previous invocation). It returns an updated set of schedules for each agent.

5) With the $UNASSIGN\_FROM\_REQUEST(request)$ function, the current agent removes any scheduled observations of *request* from its schedule and sets its frequency of observation of *request* to zero. In doing so, it frees up any overflights for other requests that may have been marked as unavailable because their time stamps conflicted with that of the agent's selected overflight for *request* (per the slew constraint).

A. From Scratch Continuous Planning

Algorithm 5: From scratch continuous planning

Input: A set of requests R, a set of agents A, and a set of overflights O.
Output: L, which is a set of schedules for each agent $a_j \in A$.

$activeRequests, withdrawnRequests, activeAgents, inactiveAgents, daySec = R, [], A, [], 86{,}400$
while $i < maxInvocations$, **do**
  for *agent inA*, **do**
    reset agent schedules and state variables (vars)
  end
  if $i > 0$, **then**
    $H_s, H_e \leftarrow H_s + daySec, H_e + daySec$
    $activeRequests, withdrawnRequests = UPDATE\_REQUESTS(activeRequests, withdrawnRequests, H_s, He)$
    $activeAgents, inactiveAgents = UPDATE\_AGENTS(activeAgents, inactiveAgents)$
  end
  $L \leftarrow SCHEDULE\_ACTIVE\_REQUESTS(activeRequests, activeAgents, O)$
end
return L

B. Incremental Continuous Planning

Algorithm 6: Incremental continuous planning

Input: A set of requests R, a set of agents A, a set of overflights O.
Output: L, a set of schedules for each agent $a_j \in A$.

```
activeRequests, withdrawnRequests, activeAgents, inactiveAgents, daySec = R, [], A, [], 86,400
while i < maxInvocations, do
  if i > 0, then
    H_s, H_e ← H_s + daySec, H_e + daySec
    activeRequests, withdrawnRequests = UPDATE_REQUESTS(activeRequests, withdrawnRequests, H_s, He)
    activeAgents, inactiveAgents = UPDATE_AGENTS(activeAgents, inactiveAgents)
    for request in withdrawnRequests, do
      for agent in activeAgents, do
        if agent previously scheduled request, then
          UNASSIGN_FROM_REQUEST(request)
        end
      end
    end
  end
  L ← SCHEDULE_NEWLY_ACTIVE_REQUESTS(activeRequests, activeAgents, O)
end
return L
```

V. Empirical Evaluation

We used grid search to tune the values for $P_{initialize}$, P_{assign}, and $P_{unassign}$ for each algorithm. We searched the range of probabilities from 0.1 to 1.0 for the combination of probabilities that performed the best on a training problem of 10 SkySats scheduling 1030 requests (634 point targets, with 66 selected for daily observation) over one week (1 August 2021 at 00:00:00 UTC to 8 August 2021 at 00:00:00 UTC) for a subset of BD algorithm variations (Table 3). Notably, the algorithms performed best with a low probability of initial assignment $P_{initialize}$ and a high probability of self-assignment P_{assign}. If fewer agents are initially self-assigned to observe a request, more may self-assign only as needed with high certainty, without agent capacities reaching n_{cap} early in execution.

Because our current results are from single-threaded simulations of distributed execution across agents, there is an $O(|A|)$ inflation in runtime. The makespan of the algorithm with parallelization should be roughly divided by $|A|$, which is the number of agents.

A. Experiment 1: Batch Planning

1. Configuration

a) The scheduling horizon was one week (1 August 2021 at 00:00:00 UTC to 8 August 2021 at 00:00:00 UTC).

b) Regarding the requests, there were 634 globally distributed terrestrial point targets (volcanos and cities), each with one observation request every 2 h of the scheduling horizon (53,256 total). Across all agents, there were nonzero overflights in the scheduling horizon for just under 35,000 of these requests, yielding the final set.

c) Regarding the agents, there were 100 in total, which were composed of Planet Labs SkySats A, B, and C 1-19, plus 79 members of various Flock constellations available on the North American Aerospace Defense Command CelesTrak[**] site as of 22 August 2022. We obtained overflight information using a calculator developed by Boerkoel et al. [9].

Table 3 Best parameter values for a subset of broadcast decentralized variations identified through grid search[a]

	$P_{initialize}$	P_{assign}	$P_{unassign}$
BD contention (sort: LFO, reward: difference)	0.1	0.9	0.6
BD contention (sort: GFO, reward: ratio)	0.1	1.0	0.2
BD contention (sort: RD, reward: ratio)	0.1	1.0	1.0
BD request satisfaction (sort: RD)	0.1	0.9	0.7

[a]Rows 1 and 4 were used for parameters in the experiments.

**http://celestrak.org/NORAD/elements/active.txt.

2. Analysis

In this batch scheduling experiment, we consider a static one-week scheduling horizon. Although the BD algorithms each run for 10 iterations within an invocation, centralized and highly decentralized are both single-pass algorithms. The performance of our broadcast decentralized algorithms is bounded above by the centralized algorithm and below by the highly decentralized algorithm. Due to the lack of coordination among agents, the highly decentralized algorithm has the highest rate of redundant observations per request and a satisfaction rate that lags behind the rest. Although the BD contention algorithm's performance plateaus around 12% below the centralized algorithm by iteration 2, BD request satisfaction continues to improve through all iterations. We attribute this to the greater stochasticity in the update procedure used by BD request satisfaction (Fig. 7). The number of broadcasts for both BD approaches is linear in time, and each broadcast is only a few words in size. Comparatively, broadcasts are minimal in the centralized and highly decentralized algorithms: one-to-many broadcasts of 1) final observation schedules and 2) the initial set of requests from the central node to the scheduling agents, respectively.

B. Experiment 2: FSCP Across All Algorithms, No Node Lossage

1. Configuration

a) The scheduling horizon was one week. It started 1 August 2021 at 00:00:00 UTC and ended 8 August 2021 at 00:00:00 UTC). Each invocation requires a one day step forward in time such that by the final invocation, the scheduling horizon was 8 August 2021 at 00:00:00 UTC to 15 August 2021 at 00:00:00 UTC.

b) Regarding the requests, there were 634 globally distributed terrestrial point targets (volcanos and cities), each with one observation request every 2 h of the scheduling horizon (53,256 total). Across all agents, there were nonzero overflights in the scheduling horizon for just under 35,000 of these requests per invocation of the scheduler, yielding the final set.

c) The agents were the same as in experiment 1.

2. Analysis

In our version of the continuous planning problem, each invocation of the scheduler corresponds to a step forward in time. We introduce further stochasticity in the set of requests by alternating two distinct sets of 10% of all requests to either stop observing or resume observing between nonzero even and odd invocations. In

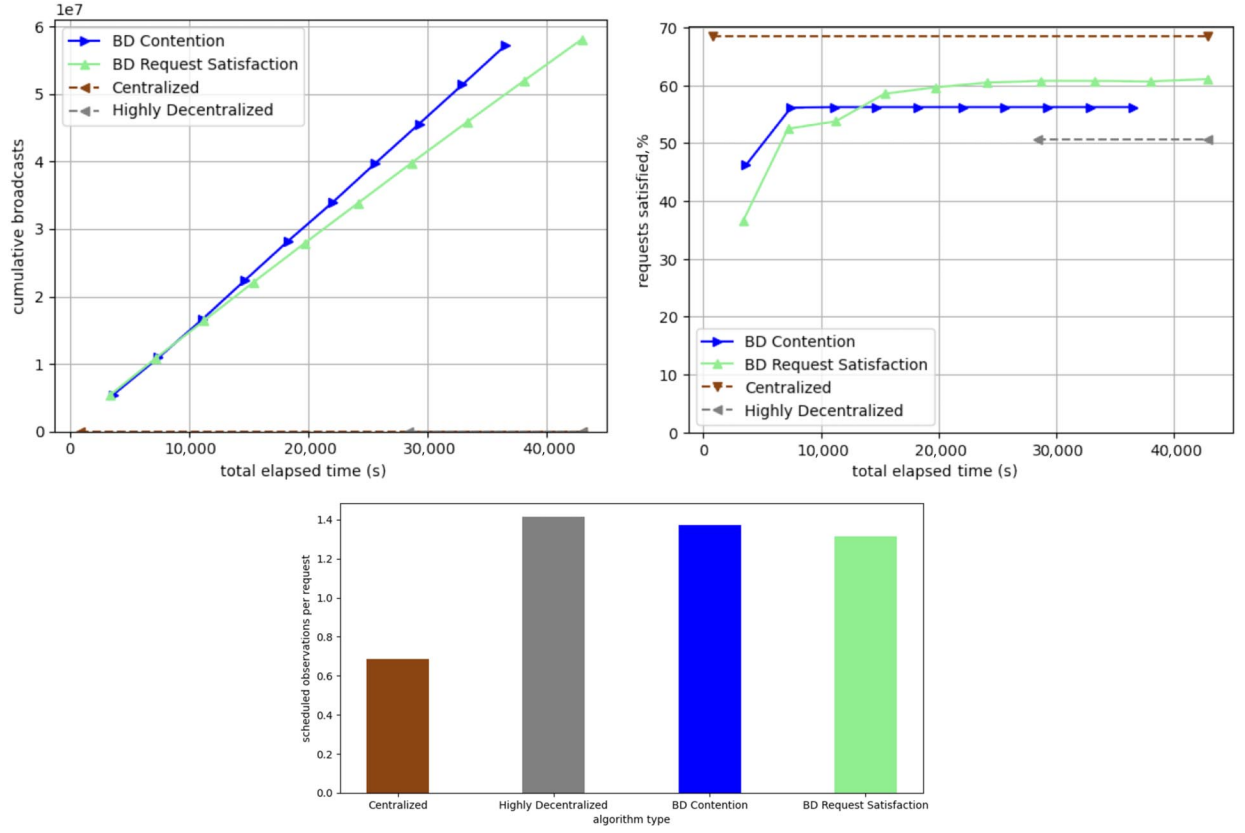

Fig. 9 Results for experiment 1: batch planning.

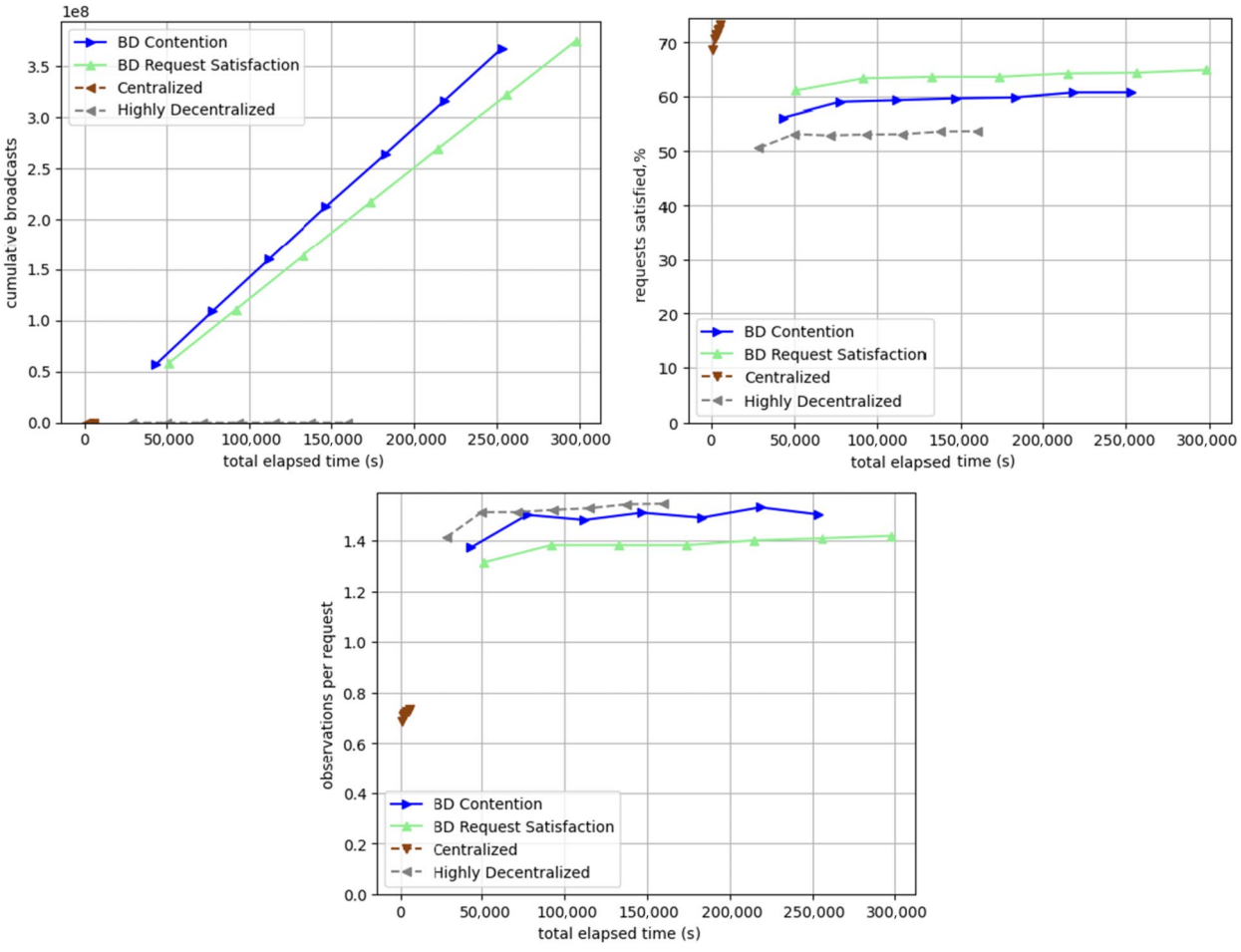

Fig. 10 Results for experiment 2: FSCP across all algorithms, with no node lossage.

FSCP, which is the subject of this experiment, the agents reschedule all requests from scratch in response to changes in the set of requests. Given the clear redundancy in rescheduling requests unaffected by a step forward in time, we note that the FSCP approach was mainly developed to make a case for incremental continuous planning, which will be discussed in the next experiment.

Each invocation of from scratch continuous planning takes roughly 50,000 s in our current nonparallelized implementation, or 0.014 s per request per agent (Fig. 9). The centralized solver is significantly faster because schedules are generated by a single scheduling node with access to all agent information. The highly decentralized algorithm, although much slower than the centralized algorithm, is also faster than our BD algorithms but has no coordination among agents, explaining its low request satisfaction rates. Again, the BD algorithms perform in between the highly decentralized and centralized algorithms with respect to request satisfaction (Fig. 10). Every algorithm's performance improves slightly between invocations of the scheduler, suggesting that the evolving set of requests had little impact on the algorithms' abilities to construct solutions from scratch. Logically, this is consistent with what we would expect for a continuous planning approach in which agents' existing schedules are not retained for modification when new requests are introduced but are discarded altogether. The average number of observations per request is fairly stable between invocations, but it exceeds a 1 : 1 ratio for all algorithms except the centralized algorithm.

C. Experiment 3: ICP Versus FSCP in BD Algorithms, No Node Lossage

1. *Configuration*

The configuration is the same as in experiment 2.

2. *Analysis*

We now evaluate both implementations of continuous planning in our broadcast decentralized algorithms (Fig. 11). As expected, because the ICP approach requires agents to modify their existing request schedules to add or remove changed observation requests, it performs roughly three times faster than its from scratch counterpart. However, the observations per request for algorithms using ICP are nearly double that of those using FSCP. In the ICP approach, when a target is "added back in" for observation in the next invocation of the scheduler, all of its observation requests in the scheduling horizon are regenerated. Agents treat these requests as entirely new and attempt initial self-assignment to them based on some initial allocation probability (Table 1). We believe that this approach adds redundancy in the number of agents that schedule observations for each request for the ICP approach. Surprisingly, BD request satisfaction outperforms BD contention on all fronts, regardless of continuous planning implementation. Given the slim distinction in the amount of information shared between the two algorithms, we attribute this to the procedure that agents use to update which requests they are self-assigned to based on the broadcast feedback from other agents (Figs. 6 and 7). Nonetheless, the performance of the BD algorithms with ICP also proves resilient to changes in the set of requests and improves over invocations; this is in part due to the high number of observations per request but also because agents continue to iterate over the same initial observation schedule, even as new requests are added and others are withdrawn.

D. Experiment 4: ICP with Node Lossage

1. *Configuration*

a) The scheduling horizon is the same as in experiment 2.

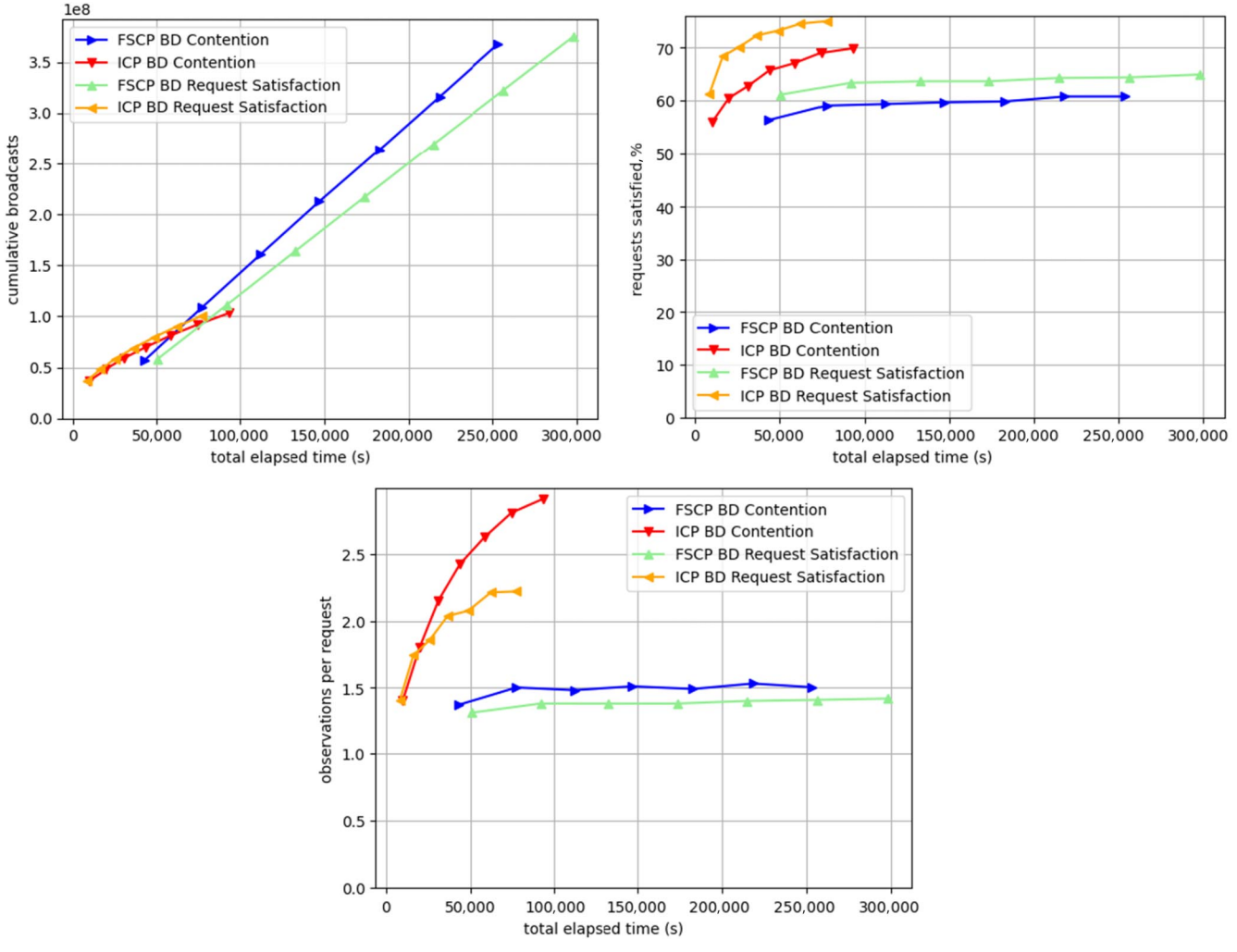

Fig. 11 Results for experiment 3: ICP vs FSCP in BD algorithms, with no node lossage.

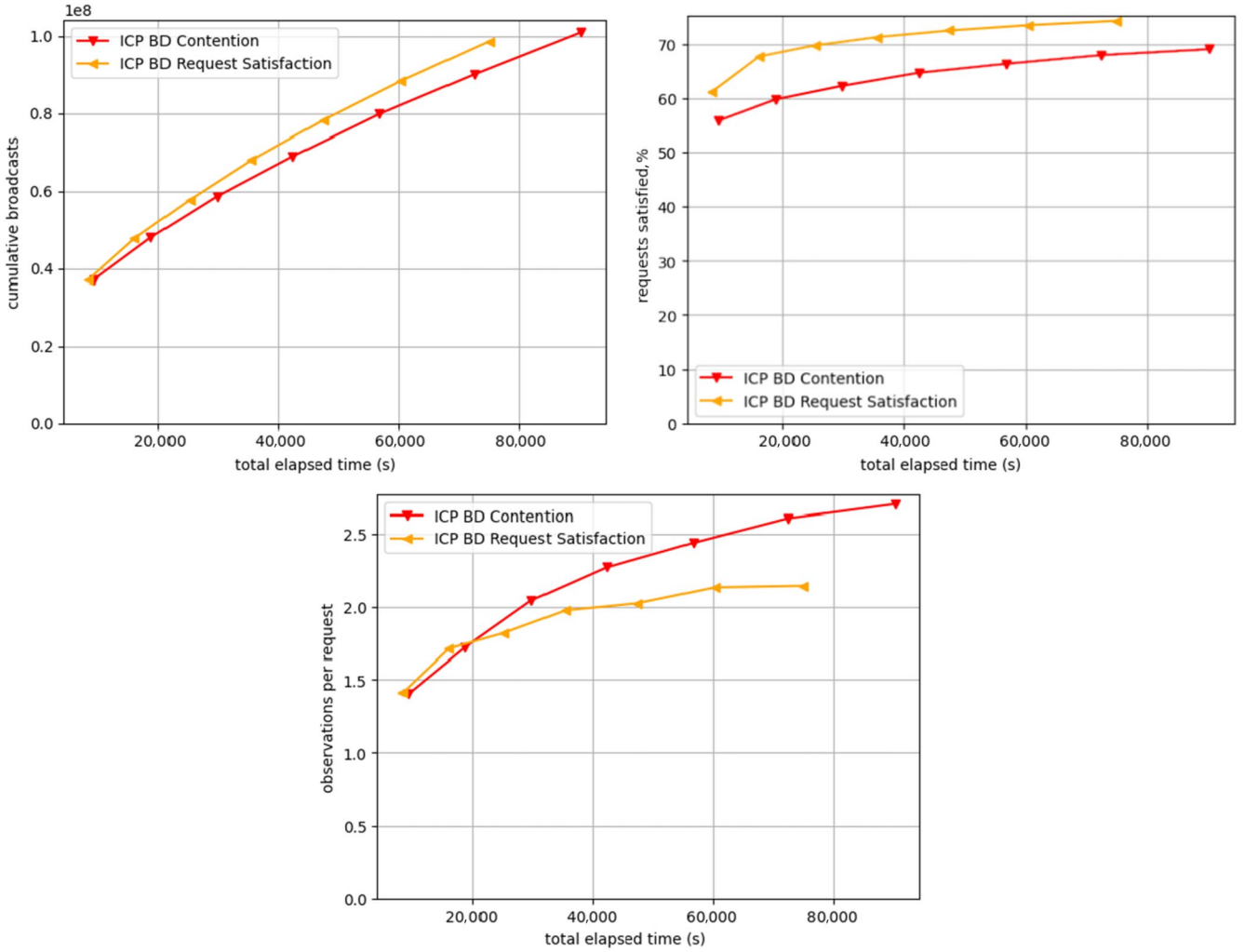

Fig. 12 Results for experiment 4: ICP with node lossage.

b) The requests are the same as in experiment 2.

c) Regarding the agents, there were 100 total, which were composed of Planet Labs SkySats A, B, and C 1-19, plus 79 members of various Flock constellations. For each invocation, 10% of the agents are withdrawn while another 10% are (re)introduced.

2. *Analysis*

This final experiment captures both elements of continuous planning studied in this paper: 1) request addition/withdrawal and 2) agent addition/loss (Fig. 12). We study these in the ICP problem for both broadcast decentralized algorithms to answer how the algorithms hold up to not only changes in the set of requests but to the disruption and addition of new nodes to the network. The addition of constellation changes seems to have little impact on agent satisfaction rates, which steadily increase across invocations. This suggests that even when agents dropped out of the network and were no longer able to schedule new requests, the remaining agents coordinated to address any coverage gaps that may have resulted.

Across all experiments, the request satisfaction implementation consistently outperforms the contention implementation of broadcast decentralized algorithms. However, because BD contention also considers whether or not a request was satisfied by other agents in the previous iteration, we would expect that BD contention would outperform BD request satisfaction. We posit that this discrepancy is attributed to the reward structure used by BD contention: the algorithm maximizes the number of overflights an agent has for a request and minimizes the number of conflicting overflights the agent has with the one it selects to satisfy the request (Table 1). When tasking spacecraft agents with observation requests, doing so rewards agents with overflights that have minimal contention instead of agents that satisfy "difficult requests" (i.e., requests with a smaller supply of agents with overflights to satisfy them). This also explains why BD contention consistently has a higher ratio of unique agent observations to requests than BD request satisfaction in our experiments. In BD request satisfaction, an agent is less likely to redundantly satisfy a request if another agent has already found an overflight to satisfy it (Fig. 7). Better understanding these reward structures and their effect upon algorithm performance is an excellent area for future work.

VI. Future Work

Further analysis of both broadcast decentralized algorithms as well as many other possible algorithms is necessary to better understand the frequency with which agents self-assign or self-unassign from requests in our experiments and to address redundancies and bottlenecks in algorithmic efficiency. Ultimately, we present one form of configuring the reward structure for BD contention that only rewards agents with minimal contention for an observation of a particular target. A compelling topic for future work is to explore other metrics for computing the reward that balance the supply of agents that can observe a particular target with the demand for observations of that target.

Additionally, we propose further development of this work by exploring the problem structure and policy variations. Toward the former, the broadcast decentralized algorithms inherit their coordination schemes from the distributed constraint optimization algorithms from which they are derived. Both the MGM algorithm and the DSA leverage constraint locality and restrict messaging to neighbors; these are agents whose decision variables are within the same cost functions, as opposed to our overly generous broadcast to

all approach [13,14]. Thus, a promising area for future work is to study what neighborhoods of agents can be defined within the large-scale constellation [i.e., grouping agents that have overflights for the same request(s)] and limit the broadcasting of state information to within those neighborhoods. Turning to geographic locality, factoring in the physical proximity of requests to one another in our scheduling (beyond our current use of a time-based slew constraint) instead of only considering temporal overflight data would allow us to study bundling requests together, as previously studied in auction-based approaches [12].

Our problem definition can also be extended to multilayer constellations beyond our current studies that focus on a single constellation in low Earth orbit (LEO). We propose incorporating multiple LEO constellations as well as constellations in medium Earth orbit and geostationary Earth orbit to enrich the solution space by prompting coordination across layers of Earth-observing satellites. This would also expand the network's ability to schedule more complex (follow-up) observations and assess/reward observation quality, depending on the asset(s) used.

VII. Conclusions

Increased space observation capabilities represent opportunities for improved space-based monitoring of a wide range of Earth science phenomena including (but not limited to) volcanos, flooding, wildfires, the cryosphere, and the biosphere. Increased in situ sensing via the IOT can also provide significant information to study these phenomena. One challenge to such developments is that of allocating observation assets. A range of centralized and decentralized methods for allocating satellite agents to observations is described.

In particular, two broadcast decentralized algorithms are outlined that implement heuristic search approaches inspired by DCOP algorithms to search for approximate solutions to the large-scale constellation request allocation problem with low data volume for agent coordination. Most coordination occurs exclusively between scheduling agents without the supervision of a central agent or the redundancy of scheduling without interagent communication. Variations of the broadcast decentralized algorithms allow users to adjust 1) the amount of information shared among agents, 2) how agents determine whether to initially self-assign to requests, 3) the order in which agents consider requests, and 4) the reward functions agents use to assess their request satisfaction.

In the current evaluation, results are presented from four experiments using realistic problem and orbit distributions to compare the current BD algorithms to their centralized and decentralized counterparts. It is demonstrates that broadcast decentralized algorithms far outperform approaches where agents schedule without any communication, demonstrating that intranetwork messaging is critical to solving the large-scale request allocation problem. Although the speed and solution quality of the current broadcast decentralized approaches trail those of an entirely centralized algorithm, through experiments with continuous planning, it is determined that the broadcast decentralized algorithms are responsive to changes in both the set of requests and the available agents without the need for or risks associated with relying on a centralized solver. When new scheduling nodes are added, the current algorithms scale without a constellationwide update required because these new nodes simply broadcast that they have picked up requests to schedule. In general, the current algorithms scale to continuous planning problems using an incremental replanning approach that can incorporate small changes in the request set into agents' existing observation schedules significantly faster than rescheduling from scratch.

Acknowledgments

Portions of this work were performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

[1] Chien, S. A., Cichy, B., Davies, A., Tran, D., Rabideau, G., Castaño, R., Sherwood, R., Mandl, D., Frye, S., Shulman, S., Jones, J., and Grosvenor, S., "An Autonomous Earth-Observing Sensorweb," *IEEE Intelligent Systems*, Vol. 20, No. 3, 2005, pp. 16–24. https://doi.org/10.1109/MIS.2005.40

[2] Chien, S. A., Mclaren, D., Doubleday, J., Tran, D., Tanpipat, V., and Chitradon, R., "Using Taskable Remote Sensing in a Sensor Web for Thailand Flood Monitoring," *Journal of Aerospace Information Systems*, Vol. 16, No. 3, 2019, pp. 107–119. https://doi.org/10.2514/1.I010672

[3] Mandl, D., Frye, S., Cappelaere, P., Handy, M., Policelli, F., Katjizeu, M.-C., Langenhove, G., Aube, G., Saulnier, J.-F., Sohlberg, R., Silva, J., Kussul, N., Skakun, S., Ungar, S., Grossman, R., and Szarzynski, J., "Use of the Earth Observing One (EO-1) Satellite for the Namibia SensorWeb Flood Early Warning Pilot," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, Vol. 6, No. 2, 2013, pp. 298–308. https://doi.org/10.1109/JSTARS.2013.2255861

[4] Chien, S. A., Davies, A. G., Doubleday, J., Tran, D. Q., Mclaren, D., Chi, W., and Maillard, A., "Automated Volcano Monitoring Using Multiple Space and Ground Sensors," *Journal of Aerospace Information Systems*, Vol. 17, No. 4, 2020, pp. 214–228. https://doi.org/10.2514/1.I010798

[5] Chien, S. A., Doubleday, J., Mclaren, D., Davies, A., Tran, D., Tanpipat, V., Akaakara, S., Ratanasuwan, A., and Mandl, D., "Space-Based Sensorweb Monitoring of Wildfires in Thailand," *Proceedings of the 2011 IEEE International Geoscience and Remote Sensing Symposium*, IEEE Press, New York, 2011, pp. 1906–1909. https://doi.org/10.1109/IGARSS.2011.6049497

[6] Branch, A., Chien, S. A., Marchetti, Y., Su, H., Wu, L., Montgomery, J., Johnson, M., Smith, B., Mandrake, L., and Tavallali, P., "Federated Scheduling of Model-Driven Observations for Earth Science," *International Workshop on Planning and Scheduling for Space (IWPSS)*, 2021, https://ai.jpl.nasa.gov/public/papers/Branch_IWPSS2021_paper_12.pdf [retrieved 05 Oct. 2022].

[7] Shah, V., Vittaldev, V., Stepan, L., and Foster, C., "Scheduling the World's Largest Earth-Observing Fleet of Medium-Resolution Imaging Satellites," *International Workshop on Planning and Scheduling for Space*, Organization for the 2019 International Workshop on Planning and Scheduling for Space, Berkeley, CA, 2019, pp. 156–161.

[8] Augenstein, S., Estanislao, A., Guere, E., and Blaes, S., "Optimal Scheduling of a Constellation of Earth-Imaging Satellites, for Maximal Data Throughput and Efficient Human Management," *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016*, edited by A. J. Coles, A. Coles, S. Edelkamp, D. Magazzeni, and S. Sanner, AAAI Press, June 2016. https://doi.org/10.1609/icaps.v26i1.13784.

[9] Boerkoel, J., Mason, J., Wang, D., Chien, S. A., and Maillard, A., "An Efficient Approach for Scheduling Imaging Tasks Across a Fleet of Satellites," *International Workshop on Planning and Scheduling for Space (IWPSS)*, 2021, https://ai.jpl.nasa.gov/public/papers/Boerkoel_IWPSS2021_paper_23.pdf [retrieved 05 Oct. 2022].

[10] Nag, S., Li, A. S., and Merrick, J. H., "Scheduling Algorithms for Rapid Imaging Using Agile Cubesat Constellations," *Advances in Space Research*, Vol. 61, No. 3, 2018, pp. 891–913. https://doi.org/10.1016/j.asr.2017.11.010

[11] Picard, G., "Auction-Based and Distributed Optimization Approaches for Scheduling Observations in Satellite Constellations with Exclusive Orbit Portions," *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, Auckland, NZ, 2022, pp. 1056–1064. https://doi.org/10.5555/3535850.3535968

[12] Phillips, S., and Parra, F., "A Case Study on Auction Based Task Allocation Algorithms in Multi Satellite Systems," AIAA Paper 2021-0185, 2021. https://doi.org/10.2514/6.2021-0185

[13] Maheswaran, R. T., Pearce, J. P., and Tambe, M., "Distributed Algorithms for DCOP: A Graphical Game-Based Approach," *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems (PDCS-2004)*, 2004, pp. 432–439.

[14] Zhang, W., Wang, G., Xing, Z., and Wittenburg, L., "Distributed Stochastic Search and Distributed Breakout: Properties, Comparison and Applications to Constraint Optimization Problems in Sensor Networks," *Artificial Intelligence*, Vol. 161, Nos. 1–2, 2005, pp. 55–87. https://doi.org/10.1016/j.artint.2004.10.004

[15] Chien, S. A., Rabideau, G., Tran, D. Q., Troesch, M., Nespoli, F., Perez-Ayucar, M., Costa-Sitja, M., Vallat, C., Geiger, B., Vallejo, F., Andres, R., Altobelli, N., and Kueppers, M., "Activity-Based Scheduling of Science Campaigns for the Rosetta Orbiter," *Journal of Aerospace*

Information Systems, Vol. 18, No. 10, 2021, pp. 711–727. https://doi.org/10.2514/1.I010899

[16] Maillard, A., Jorritsma, M., and Schaffer, S., "Sailing Towards an Expressive Scheduling Language for Europa Clipper," *Knowledge Engineering for Planning and Scheduling (KEPS), International Conference on Automated Planning and Scheduling (ICAPS KEPS)*, 2021, https://ai.jpl.nasa.gov/public/papers/Maillard_KEPS2021_paper_22.pdf [retrieved 05 Oct. 2022].

[17] Chien, S., Johnston, M., Policella, N., Frank, J., Lenzen, C., Giuliano, M., and Kavelaars, A., "A Generalized Timeline Representation, Services, and Interface for Automating Space Mission Operations," *International Conference on Space Operations (SpaceOps 2012)*, AIAA, Reston, VA, 2012. https://doi.org/10.2514/6.2012-1275459

[18] Joslin, D. E., and Clements, D. P., "Squeaky Wheel Optimization," *Journal of Artificial Intelligence Research*, Vol. 10, May 1999, pp. 353–373. https://doi.org/10.1613/jair.561

[19] Fioretto, F., Pontelli, E., and Yeoh, W., "Distributed Constraint Optimization Problems and Applications: A Survey," *Journal of Artificial Intelligence Research*, Vol. 61, March 2018, pp. 623–698. https://doi.org/10.1613/jair.5565

E. Atkins
Associate Editor