Journal of Artificial Intelligence Research 82 (2025) 169-208 Submitted 08/2024; published 01/2025

Decentralized, Decomposition-Based Observation Scheduling for a Large-Scale Satellite Constellation

Itai Zilberstein ITAI.M.ZILBERSTEIN@JPL.NASA.GOV
Jet Propulsion Laboratory, California Institute of Technology,
Pasadena, CA 91011, USA

Ananya Rao ANANYARA@ANDREW.CMU.EDU
Jet Propulsion Laboratory, California Institute of Technology,
Pasadena, CA 91011, USA
Carnegie Mellon University,
Pittsburgh, PA 15213, USA

Matthew Salis MATTHEW.SALIS@JPL.NASA.GOV
Steve Chien STEVE.A.CHIEN@JPL.NASA.GOV
Jet Propulsion Laboratory, California Institute of Technology,
Pasadena, CA 91011, USA

Abstract

Deploying multi-satellite constellations for Earth observation requires coordinating potentially hundreds of spacecraft. With increasing onboard capability for autonomy, we can view the constellation as a multi-agent system (MAS) and employ decentralized scheduling solutions. We analyze the multi-satellite constellation observation scheduling problem (COSP) and formulate it as a distributed constraint optimization problem (DCOP). COSP requires scalable inter-agent communication and computation and consists of millions of variables which, coupled with the assumptions and structure, make existing DCOP algorithms inadequate for this application. We develop a scheduling approach that employs a carefully constructed heuristic, referred to as the Geometric Neighborhood Decomposition (GND) heuristic, to decompose the global DCOP into sub-problems to enable the application of DCOP techniques. We present the Neighborhood Stochastic Search (NSS) algorithm, a decentralized algorithm to effectively solve COSP and other large-scale distributed problems, using decomposition. The experiments confirm the efficacy of the approach against baseline algorithms, and we discuss the generality of NSS, GND, and properties of COSP to other domains.

1. Introduction

Large-scale, Earth-observing satellite constellations with hundreds of spacecraft are becoming increasingly prominent in order to monitor Earth phenomena. The first satellite constellations started with just a couple spacecraft (e.g. Maxar, Spot Image). More recently, constellations have grown to operate with tens of spacecraft (e.g. SkySat, IceEye, Capella, Satellogic). With reduced launch costs, constellations are trending towards even larger scales with hundreds or thousands of spacecraft (e.g. Spire, Canon, SatRev, Spacety, Planet Lab's Dove, and the Space Development Agency) (NewSpace, 2023). Observation scheduling for a large-scale constellation requires fusing information from many sources and

tasking space assets with varying constraints, capabilities, and visibility of Earth targets. In addition to Earth observation, satellites are deployed for a variety of applications, including forming large internet constellations (SpaceX, 2023). Any multi-satellite constellation that requires coordinating agents poses a challenging planning and scheduling problem.

In practice, satellite observation scheduling is typically done in a centralized fashion, in which a single controller develops a single schedule that specifies the actions of every satellite (Shah et al., 2019). Even most research efforts focus on centralized solutions (Boerkoel et al., 2021; Nag et al., 2018). While centralized approaches can provide high-quality solutions, reliance on a single computing node makes them vulnerable to single-point failures and can increase the communications burden when the spacecraft state constantly needs to be monitored and shared with the controller. Many centralized approaches rely on local search or divide-and-conquer algorithms, making distributed solutions natural. In addition, scheduling for satellite constellations may require handling shifting objectives or a dynamically changing problem. Satellites leave and join the constellation, and requests are added and removed from the scheduling campaign. For example, commercial satellite constellations typically deploy portions of a larger constellation progressively over time. We can also aggregate multiple constellations into one *federated* observation system, which inherently requires distributed control when spacecraft are operated by multiple institutions. The increasing number of constellation providers opens the door to larger and larger federated systems.

We also desire satellites that are capable of critical tasking, such as tracking a natural disaster, which requires real-time response. New onboard capabilities for edge computing enable data analysis onboard, meaning dynamic, rapid decision-making can happen without ground control on much faster timelines (Chien et al., 2005; Kangaslahti et al., 2024; Chien et al., 2024; Zilberstein et al., 2024a). Other commercial applications, such as Planet's SkySat, require continuous planning to provide timely completion of user observation requests (Planet, 2023). Re-scheduling for an entire large-scale constellation is computationally intensive and may be required at an unrealistic frequency for a single computing source. All these factors point towards decentralized, onboard control.

Treating the constellation as a multi-agent system (MAS) enables the application of decentralized scheduling solutions. Decentralized scheduling addresses both the system's robustness and the vulnerabilities of a central controller (Bonnet & Tessier, 2008; Phillips & Parra, 2021). Many MAS problems are designed to optimize a global cost function in which agents control the parameters of the function and must communicate to coordinate their parameter assignments (Dorri, Kanhere, & Jurdak, 2018).

In many applications, computational resources are limited and communication may be unreliable for large message volumes. Satellites are typically constrained by RAM and CPU cycles, and share these resources with other, critical flight software, requiring any scheduling software to be lightweight. While the capabilities of onboard computation have been increasing, any exponential cost search is still infeasible, especially for scheduling applications that require continual re-planning (Clement et al., 2007; Swope et al., 2023; Dunkel et al., 2023). When it comes to communication, satellite cross-links are expensive and not always maintainable. For example, orbiters around a comet or the sun may experience communication interference or infrequent lines of sight with each other. For an Earth-orbiting constellation, satellites have limited cross-link capability. Bandwidth limits the rate of com-

munication, and visibility between two spacecraft constrains the duration of a transmission (Lin et al., 2024). For this reason, we desire algorithms that procure a limited amount of messaging.

We present the *multi-satellite constellation observation scheduling problem* (COSP), as well as formulate the problem as a distributed constraint optimization problem (DCOP). The assumptions of COSP differ from those of a DCOP, making current DCOP solvers inadequate. COSP assumes agents are unaware of the capabilities of other agents, yet know their existence. In contrast to DCOPs, agents typically know these capabilities, and this knowledge is key to many solution techniques. This assumption also does not hold for federated systems operated by multiple institutions. In addition, DCOP solutions tend to rely heavily on computation and communication, however, a desired decentralized solution of COSP is both effective in solution quality and cheap in computation and communication.

COSP requires agents to coordinate the assignments of millions of variables, and do so within stringent time, memory, and messaging constraints. These variables are determined by the observation requests of the system and the satellite overflights of requested targets. Most satellites can satisfy most requests on multiple occasions leading to the total number of variables being polynomial in the number of agents and requests. By decomposing the global problem, we can employ DCOP techniques to solve smaller sub-problems, producing global solutions more efficiently. We construct a heuristic, called the *Geometric Neighborhood Decomposition* (GND) heuristic, that partitions the agents and requests in a coordinated fashion to instantiate sub-problems that are advantageous to solve. Each agent individually computes the heuristic using only knowledge of the requests to schedule and the configuration of the constellation. GND addresses the assumptions of COSP and enables efficient, high-quality scheduling despite limited knowledge through aggregated estimation of agent capabilities.

Satellite constellations are typically designed to optimize the supply (e.g. volume of satellite capability to view a target) given the expected demand (e.g. how much a target is expected to be requested for observation) for a target set (Lo, 1999). This couples the scheduling problem with the constellation configuration and can be solved as another search problem (Schaffer et al., 2018). Constellations are typically fixed relative to the number of scheduling runs they execute. GND is grounded in geometric computation and designed to leverage this supply and demand relationship. GND is composed of three layers that partition the agents and requests into sub-problems. The goal is for the constellation to maximize the number of requests satisfied while adhering to downlinks and memory constraints.

GND is parameterized such that the sub-problems produced can be of arbitrary size. Through this decomposition, we can deploy DCOP algorithms on constant-sized sub-problems rather than the global problem that scales with the number of agents and requests. To solve each sub-problem, we build on the *Broadcast Decentralized* algorithm (BD) (Parjan & Chien, 2023), an adaptation of two incomplete DCOP algorithms, *Maximum Gain Messaging* (MGM) (Maheswaran et al., 2004) and *Distributed Stochastic Search Algorithm* (DSA) (Zhang et al., 2005), for the application of multi-satellite observation scheduling. We refer to our developed algorithm as *Neighborhood Stochastic Search* (NSS). Our algorithm extends the BD algorithm in two major aspects: (1) it is scalable to large problem instances

in both computational complexity and communication complexity and (2) it enables constraint reasoning, such as resource-constrained scheduling.

Empirical results demonstrate the efficacy of our approach on small and large problem instances compared to decentralized and centralized baselines. On small problem instances, we show the gap to optimal solutions, while large problem instances enforce the performance at scale, including run-time and message volume results. We aim to close the gap between decentralized solutions and centralized solutions while precluding an infeasible amount of computation and messaging. Our contributions are:

1. uncovering the obstacles in applying existing DCOP techniques to the large-scale, multi-satellite constellation observation scheduling problem (COSP),

2. introducing a decomposition-based heuristic approach to solve the scheduling problem and presenting the NSS algorithm which utilizes the decomposition scheme, and

3. demonstrating the efficacy of our approach on realistic problem instances.

This article extends on a previously published conference paper (Zilberstein et al., 2024b). We provide a more in-depth discussion, extended theoretical analysis, broader experimental results and analysis, and improvements and variations of the initial algorithms and heuristics published.

2. Related Work

There are many aspects of satellite observation planning ranging from visibility computation, to downlink scheduling, to constraint-based task allocation. This paper is mostly concerned with the last subject. Previous work has predominantly focused on centralized solutions to the multi-satellite, resource-constrained scheduling problem (Globus et al., 2004; Augenstein et al., 2016; Nag et al., 2018; He et al., 2018; Shah et al., 2019; Wang et al., 2020; Squillaci et al., 2021; Boerkoel et al., 2021; Eddy & Kochenderfer, 2021; Squillaci et al., 2023; Chatterjee & Tharmarasa, 2024).

More recently, decentralized scheduling approaches have proposed auction-based methods (Picard, 2021; Phillips & Parra, 2021) and heuristic search-based methods relying on broadcasting or epidemic protocols (Parjan & Chien, 2023; Bonnet & Tessier, 2007, 2008).

The auction-based methods rely on a centralized controller to act as an auctioneer and have a prohibitive communication and computational complexity. Each agent exchanges a polynomial (in the requests) number of messages with the auctioneer. Removing the central auctioneer is possible, but results in an explosion of the communication complexity depending on the network topology.

We build on the work from Parjan and Chien, which attempts to address some of the limitations of the auction-based methods; specifically having a centralized auctioneer. In their approach, called BD, each agent uses globally communicated satisfaction information as a search heuristic. The main limitation is that this approach requires each agent to send a high volume of messages to every other agent, resulting in a communication complexity at each iteration that is polynomial in the number of agents and requests. Finally, Bonnet and Tessier propose an approach in which agents continuously communicate to form coalitions, or groups of agents to schedule specific tasks. The major drawback of this approach is its

potential to scale. The authors evaluated scenarios with less than ten agents. In a constellation with hundreds or thousands of agents, there will likely be large discrepancies in agents' believed coalitions, degrading the solution quality. The authors also provide no bound on the required communication. All the decentralized approaches discussed above rely heavily on agent computation and communication (at least polynomial in the number of agents and requests) and most approaches were only evaluated on small problem instances (tens of agents). In general, decentralized solutions trade off quality for distributed computation and control.

Multi-satellite observation scheduling has been framed as a DCOP previously (Picard, 2021; Parjan & Chien, 2023). DCOP algorithms tend to suffer from significant computational complexity or a large reliance on agent messaging. Solving a DCOP optimally is known to be NP-Hard (Modi et al., 2005). Complete algorithms, such as *SyncBB* (Hirayama & Yokoo, 1997), *ADOPT* (Modi et al., 2005), *ConcFB* (Netzer, Grubshtein, & Meisels, 2012), *AFB* (Gershman, Meisels, & Zivan, 2009), *DPOP* (Petcu & Faltings, 2005), or *OptAPO* (Mailler & Lesser, 2004) are computationally infeasible for our problem scale. Many complete algorithms are distributed variations of a centralized counterpart (Fioretto, Pontelli, & Yeoh, 2018). We employ a centralized branch and bound algorithm to find optimal solutions on small problem instances. However, the computational cost of the branch and bound approach prevents finding optimal solutions on large problem instances. Any complete DCOP algorithm would suffice for small instances, but would suffer the same scalability issues as the centralized branch and bound, making them impracticable for the application.

On the other hand, incomplete DCOP algorithms, such as *Max-Sum* (Stranders et al., 2009), *Maximum-Gain Messaging* (MGM) (Maheswaran et al., 2004), *Distributed Stochastic Search* (DSA) (Zhang et al., 2005), or *Distributed Gibbs* (D-Gibbs) (Nguyen et al., 2019), which trade off optimality for scalability, still require notable messaging and computation. *Max-Sum*, an inference-based method, can provide quality guarantees on the solution, but incurs exponential cost. D-Gibbs is a sampling-based method that approximates the solution without quality guarantees. MGM and DSA are two search algorithms that are the foundation of our scheduling algorithm, *Neighborhood Stochastic Search* (NSS). MGM and DSA perform local search to iteratively improve the global solution. Genetic algorithms, such as AED, have also been applied to DCOPs, incurring similar computation and communication complexity as MGM and DSA (Mahmud et al., 2019). Improving the efficiency of incomplete DCOP algorithms has been a focus of recent efforts (Khan et al., 2018a; Deng & An, 2020; Cohen et al., 2020; Zivan et al., 2017).

NSS is motivated by *Region-optimal* algorithms (Pearce & Tambe, 2007). These algorithms solve sub-problems optimally, reducing the cost of complete algorithms, albeit to the size of sub-problems. NSS extends *Region-optimal* algorithms by obtaining incomplete solutions to sub-problems, reducing the complexity when sub-problems remain large. Another divide-and-conquer method for solving DCOPs is the application of the distributed large neighborhood search (DLNS) framework to DCOPs (Hoang et al., 2018). This approach heuristically selects regions of the search space to refine, iteratively improving the global solution. Large neighborhood search was used to construct centralized solutions to the multi-satellite scheduling problem, albeit for smaller problem instances (Squillaci et al., 2023). While DLNS can rely heavily on agent communication, directly applying

Algorithm	Computation	Communication	Source
ADOPT	$O(2^n)$	$O(2^n)$	(Modi et al., 2005)
AFB	$O(2^n)$	$O(2^n)$	(Gershman et al., 2009)
ConcFB	$O(2^n)$	$O(2^n)$	(Netzer et al., 2012)
DPOP	$O(2^w)$	$O(n{\cdot}2^w)$	(Petcu & Faltings, 2005)
OptAPO	$O(2^n)$	$O(2^n)$	(Mailler & Lesser, 2004)
SyncBB	$O(2^n)$	$O(2^n)$	(Hirayama & Yokoo, 1997)
Region-Optimal	$O(k{\cdot}2^w)$	$O(k{\cdot}n{\cdot}2^w)$	(Pearce & Tambe, 2007)
Max-Sum	$O(k{\cdot}2^m)$	$O(k{\cdot}m)$	(Stranders et al., 2009)
AED	$O(k{\cdot}m^2)$	$O(k{\cdot}m^2)$	(Mahmud et al., 2019)
DLNS	$O(k{\cdot}n)$	$O(k{\cdot}n)$	(Hoang et al., 2018)
D-Gibbs	$O(k{\cdot}m)$	$O(k{\cdot}m)$	(Nguyen et al., 2019)
MGM	$O(k{\cdot}m)$	$O(k{\cdot}m)$	(Maheswaran et al., 2004)
DSA	$O(k{\cdot}m)$	$O(k{\cdot}m)$	(Zhang et al., 2005)
BD	$O(k{\cdot}n)$	$O(k{\cdot}n)$	(Parjan & Chien, 2023)

Table 1: Complexity of DCOP algorithms applied to COSP shown per agent. n = number of variables, m = max number of neighbors, k = iterations, w = induced width of pseudo-tree. We assume for simplicity that variables are one-to-one per agent and variable domains are binary.

the framework to the decentralized multi-satellite scheduling problem may be a promising future endeavor. Some DCOP settings assume that there is not a fixed mapping of variables to agents, and a mapping can be dynamically computed. The *Mapping of Nodes to the participating Agents heuristic* (MNA) can be leveraged as a decomposition technique by computing sub-mappings of agents to nodes for divide-and-conquer DCOP algorithms (Khan et al., 2018b).

We summarize the complexity of these DCOP algorithms in Table 1. In the table, n denotes the number of variables, m denotes the max number of neighbors, w denotes the *induced width* of the pseudo-tree, and k denotes the number of iterations for relevant algorithms. We assume that variable domains are binary and there is only one variable per agent. However, COSP instances are multi-variable and we typically have $m = O(n)$ and $w = O(n)$, therefore algorithms such as *Max-Sum* are exponential in n.

In the next sections, we discuss the challenges of applying these algorithms to the multi-satellite constellation observation scheduling problem (COSP). We then show how heuristically decomposing the problem can overcome the hurdles in their deployment while still providing high-quality solutions.

3. Problem Formulation

In this section, we outline the *multi-satellite constellation observation scheduling problem* (COSP). The main application of COSP is for Earth observation. However, the formulation extends to orbits around other bodies. We start by formally defining the problem. Then,

we present the problem as a DCOP and discuss some theoretical properties of the problem, including the barriers to applying current DCOP algorithms. Similar formulations have been examined in previous work (Picard, 2021; Parjan & Chien, 2023).

3.1 Multi-Satellite Constellation Observation Scheduling Problem

We formally define COSP in the following way.

Definition 3.1 (Multi-Satellite Constellation Observation Scheduling Problem)**.** *The multi-satellite constellation observation scheduling problem (COSP) is a* 6*-tuple* $\langle H, A, R, \mathcal{X}, D, C \rangle$*. The main components are*

- H*: the scheduling horizon,*
- A*: the set of agents,*
- R*: the set of observation requests,*
- $\mathcal{X}$*: the set of variables,*
- D*: the set of downlinks, and*
- C*: the set of constraints.*

Each variable is controlled by an agent and represents a task that can be scheduled to satisfy a single request in R*. The goal of the optimization problem is to maximize the number of requests satisfied while not violating the constraints of any agent.*

We provide more detail on the main components and sub-components below.

1. $H = [h_s, h_e]$: the scheduling horizon, starting at time h_s and ending at time h_e.
2. $\mathcal{K}$: the set of orbital planes. An orbital plane defines the geometric plane that contains a collection of satellite orbits. We denote $K \in \mathcal{K}$ as an orbital plane, and $k \in K$ for the specific orbit of a satellite within that plane. The number of satellites in an orbital plane is denoted as $|K|$. Figure 1 shows the orbits defined by a single orbital plane with 5 satellites. The satellites are shown as blue dots.
3. A: the set of agents. Each $a_i \in A$ is an individual satellite in the constellation. We define $a_i = (k, m)$ where k is the orbit of the satellite and $m \in \mathbb{R}^+$ is the memory capacity. We use the notation $k(a_i)$ and $m(a_i)$ to denote these values for agent a_i, and will use the same notation for equivalent indexing.
4. T: the set of point targets on Earth. Each $t_i \in T$ is defined as $t_i = (lat,\ lon)$ where lat is the latitude and lon is the longitude of the target.
5. R: the set of requests. Each $r_i \in R$ is defined as $r_i = (t, h)$ which denotes the target to observe, $t \in T$, and when in the scheduling horizon to observe, $h \subset H$. Note, h could be both a single interval or a set of disjoint intervals. A request is satisfied when a single observation is taken of t within the time interval h.

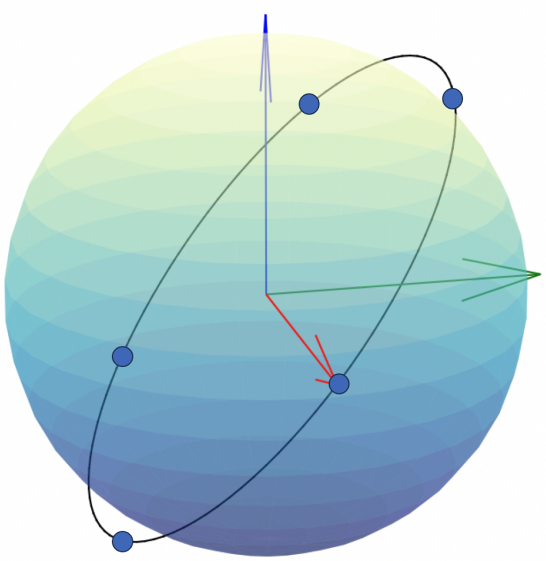

Figure 1: Visualization of an orbital plane with 5 satellites.

6. For each agent, we define the following sets that compose $\mathcal{X}, D$, and C respectively.

   (a) S_{a_i}: the set of possible *request fulfillments* for agent a_i. A request fulfillment, $s_j \in S_{a_i}$, is a task that can be scheduled to satisfy a request. From R and $k(a_i)$, we derive visibilities for each target and each satellite. Sub-intervals of these visibilities are used to create the tasks that define the fulfillments for satellite a_i to satisfy request r_j. We define $s_j = (r, h, m)$ where $r \in R$ is the request being satisfied, $h \subset h(r)$ is the interval of the observation (including processing and slewing time), and $m \in \mathbb{R}^+$ is the amount of memory required to take the observation. Note that the memory required may vary across agents if they utilize different sensors or image-processing. There may be zero, one, or multiple opportunities for an agent to satisfy any request.

   (b) $\mathcal{X}_{a_i}$: the set of Boolean decision variables for agent a_i. For each $s_j \in S_{a_i}$ we define the Boolean decision variable $x_j \in \mathcal{X}_{a_i}$ where $x_j = 1 \iff$ agent a_i schedules task s_j. We denote $x(s_j) = x_j$ for agent a_i.

   (c) D_{a_i}: the set of downlinks for agent a_i. A downlink, $d_j \in D_{a_i}$, is defined as $d_j = (h, m)$ where $m \in \mathbb{R}^+$ is the maximum amount of data that can be downlinked, and the interval $h \subset H$ is the time window for the downlink. We assume that no possible tasks occur during a downlink, and all downlinks are mandatory.

   (d) $C_{a_i} = C_{D_{a_i}} \cup C_{S_{a_i}}$: the set of constraints where

$$C_{D_{a_i}} = \bigcup_{d_j \in D_{a_i}} c_{d_j}$$

   and

$$c_{d_j} = \sum_{s_l \in S_{a_i}^{d_j}} x(s_l) \cdot m(s_l) \leqslant \text{MIN}(m(a_i), m(d_j)).$$

Here, $S_{a_i}^{d_j}$ denotes the set of possible tasks for which the soonest downlink window in the future is d_j. This constraint enforces that agent a_i never exceeds its memory capacity and that for any observations it takes, it will be able to downlink the data at the soonest opportunity. We define

$$C_{S_{a_i}} = \bigcup_{s_j, s_l \in S_{a_i}} c_{s_j, s_l}$$

where

$$c_{s_j,s_l} = \left[x(s_j) \cdot x(s_l) + \mathbb{I}(h(s_j) \cap h(s_l) \neq \varnothing) \leqslant 1\right].$$

This constraint ensures that no tasks are scheduled to overlap. Here, $\mathbb{I}$ denotes the indicator function that is 1 if the input condition is true and 0 otherwise. The constraint that no task is performed during a downlink is not explicitly modeled with a constraint, but it is enforced during task construction which is discussed later.

The goal of the optimization problem is to maximize the number of requests satisfied while not violating the constraints of any agent. We define a solution, X, of COSP.

Definition 3.2. *A solution to COSP, X, is the assignment of each $x \in \mathcal{X}$ such that all constraints in C are satisfied.*

Note that in this formulation, the downlink model is a simplification. We assume that all agents must take every downlink opportunity and cannot fill their memory past what they will be able to off-board at the next available downlink. We also assume that ground stations can service multiple downlinks simultaneously. In reality, ground stations have limited antennas, and therefore not all agents can be serviced at all opportunities. Agents could also keep memory onboard to downlink at a future opportunity. Downlink scheduling is another challenging decentralized scheduling problem in its own right, but is beyond the scope of this work. Future work would examine ways to optimize downlink operations as part of observation planning. Despite this simplification, the general definition of COSP can model more complex formulations of downlinking.

3.2 DCOP Formulation of COSP

We can formulate COSP as a distributed constraint optimization problem (DCOP). A DCOP is a five-tuple $\langle A, \mathcal{X}, \mathcal{D}, \mathcal{F}, \alpha \rangle$ which we define below.

- A: the set of agents.
- $\mathcal{X}$: the set of variables.
- $\mathcal{D}$: the set of finite variable domains, where $\mathcal{D}_i$ is the domain of variable $x_i \in \mathcal{X}$.
- $\mathcal{F} = \bigcup_{\mathcal{S} \subseteq \mathcal{X}} \left[\times_{x_i \in \mathcal{S}} \mathcal{D}_i \to \mathbb{R}\right]$: the set of utility functions (sometimes called constraints) where each $f \in \mathcal{F}$ takes in an input $\mathcal{S} \subseteq \mathcal{X}$ and outputs a utility based on their assignments. Given an assignment of the variables, X, we use $X^{\mathcal{S}}$ to denote the relevant input of a utility function.

- $\alpha : \mathcal{X} \rightarrow A$ is the mapping function of variables to agents. $\alpha(x) \mapsto a$ denotes that agent a controls the assignment of variable x.

The goal of a DCOP is to obtain an assignment of all variables as to maximize (or minimize) the sum of the utility functions,

$$X^* = \text{argmax}_X \sum_{f \in \mathcal{F}} f\left(X^{\mathcal{S}}\right).$$

We can define the above for our problem:

- A: the set of satellites as previously defined.
- $\mathcal{X} = \bigcup_{a_i \in A} \mathcal{X}_{a_i}$: the set of Boolean decision variables for every agent's possible task set as previously defined.
- $\mathcal{D} = \bigcup_{x \in \mathcal{X}} \{0, 1\}$: all variable domains are Boolean.
- $\mathcal{F}(X) = \bigcup_{a_i \in A} f_{a_i}(X_{a_i}) \cup \bigcup_{r_j \in R} f_{r_j}(X_{r_j})$ where

$$f_{a_i}(X_{a_i}) = \begin{cases} 0 & \text{if } X_{a_i} \text{ satisfies } C_{a_i} \\ -\infty & \text{else} \end{cases}$$

and

$$f_{r_j}(X_{r_j}) = 1 - \prod_{x \in X_{r_j}} (1 - x).$$

Here, X_{a_i} is the set of variables in the solution, X, such that $\alpha(x) = a_i$, and X_{r_j} is the set of variables such that $x = x(s_l)$ and $r(s_l) = r_j$. See that $f_{r_j}(X_{r_j}) = 1$ iff there exists a satellite satisfying request r_j and $f_{a_i}(X_{a_i}) = 0$ iff agent a_i has a schedule that satisfies its constraints.

- $\alpha(x) = a_i \iff x \in \mathcal{X}_{a_i}$ maps a variable to the agent that can schedule the associating request fulfillment.

In the standard definition of a DCOP, one agent controls exactly one variable. However, COSP requires one agent to control many variables. We use a similar paradigm to the *Multi-Variable Agent Decomposition framework* (MVA) (Fioretto, Yeoh, & Pontelli, 2016) in which we consider both a global problem (coordinating the schedules across agents) and a local problem (developing the schedule of a single agent). MVA is quite natural for COSP since agents need to coordinate global satisfaction of requests, while each individually managing resource constraints. This is apparent in the reward function structure; the set of f_{r_j} determines the global problem while each f_{a_i} determines an agent's local problem.

3.3 Challenges with DCOP Solutions

While the problem can be represented as a DCOP, there exist roadblocks to applying existing DCOP algorithms. The constraint graph defined by this formulation has a minimum of $|A| + |R|$ complete sub-graphs derived from f_{a_i} and f_{r_j}. Each $a_i \in A$ contributes a clique

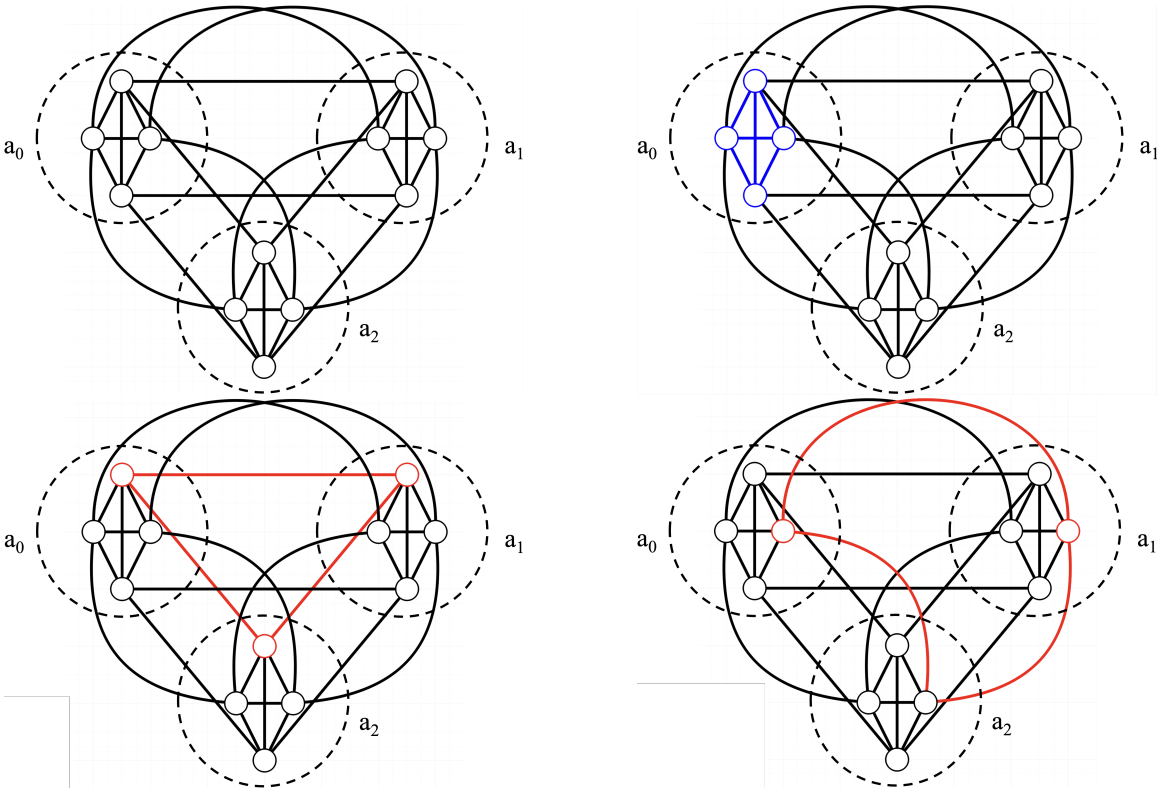

Figure 2: Example constraint graph with $|A| = 3$ and $|R| = 4$ (top left). Nodes represent a request fulfillment for agent a_i, and edges denote a shared constraint among variables. Dotted circles highlight which agents control which variables. Top right highlights the clique induced by f_{a_0}. The bottom two graphs highlight cliques induced by two different f_{r_j}.

of size $|\mathcal{X}_{a_i}|$, and each $r_j \in R$ contributes a clique of size $|\mathcal{X}_{r_j}|$. In many problem instances, request time windows are long enough such that the majority of agents are able to satisfy any particular request. This results in $|\mathcal{X}_{a_i}| = \Omega(|R|)$ and $|\mathcal{X}_{r_j}| = \Omega(|A|)$. In addition, most of these cliques are highly connected to each other, resulting in a cyclic graph. Figure 2 shows an example of the structure we discuss for a problem instance with 3 agents and 4 requests, including highlights of the cliques induced by different reward functions.

Recall that we consider constellations with hundreds of agents and thousands of requests. Therefore, since each agent a_i controls all variables in $\mathcal{X}_{a_i}$, the neighborhood of variables for an *agent* is $\Omega(|A| \cdot |R|)$, which is on the order of 10^5 variables. The problem scale coupled with the structure makes it unsuitable for many DCOP algorithms. Complete DCOP algorithms, such as *SyncBB* (Hirayama & Yokoo, 1997) or *ADOPT* (Modi et al., 2005), are simply infeasible for the problem size, having a computational complexity of $O(2^{|A|\cdot|R|})$, which is on the order of 10^{30102}. Incomplete DCOP algorithms, such as *Max-Sum* (Stranders

Problem Model	Variables Per Agent	Known Agents	Known Constraints
DCOP	1	Neighboring Agents	Neighboring Agents
COSP	Many	All Agents	Self Only

Table 2: Differences between DCOP assumptions and COSP assumptions.

et al., 2009) or *Maximum Gain Messaging* (MGM) (Maheswaran et al., 2004), require many iterations, in which at every iteration, each agent would exchange $\Omega(|A|)$ messages, each with size $\Omega(|R|)$, which is on the order of 10^7 total volume (Fioretto et al., 2018). Finally, we mention that the DLNS and region-optimal approaches require defining sub-problems. One application of our approach is outlining methods for sub-problem selection. However, the above two algorithms still incur substantial costs when sub-problems remain large. As mentioned previously, the MNA heuristic (Khan et al., 2018b) can be leveraged to create sub-problems for DCOPs. We note that in COSP, MNA will require every agent to communicate with every other agent. In addition, node-to-agent mappings typically require agents to have local constraint graph knowledge which is not assumed.

Besides computation, we discuss one nuance of COSP that makes COSP a harder problem than a traditional DCOP. In standard DCOPs, agents know the variables and constraints of neighboring agents (Fioretto et al., 2018). An agent is unaware of the *values* of other agents' variables, but is privy to their existence. In COSP, we assume that agents are aware of the existence of other *agents*, but have no knowledge of the request fulfillments (variables) other agents are attempting to schedule. The implication is that an agent does not know the local structure of the constraint graph for which it is a part. Note that it takes one broadcast for each agent to share *all* their request fulfillments before the problem becomes a standard DCOP. However, this requires an enormous amount of communication (and memory) and is undesirable. This nuance is motivated by the requirements of the application, and our solutions address this need. In contrast, an assumption of DCOP algorithms is that agents can use the known local topology of the constraint graph to form communication channels. This nuance is another reason why previous DCOP algorithms are not directly suitable for COSP. Table 2 summarizes the differences in assumptions between a DCOP and COSP. The variables per agent is one for traditional DCOPs, but as mentioned previously frameworks such as MVA exist to handle multiple variables per agent.

The structure of COSP encompasses a class of multi-agent scheduling problems and other distributed coordination problems. The basic elements of this class of problems are

- A: a set of agents,
- R: a set of tasks/requests/goals to be satisfied, and
- C: a set of constraints.

These problems capture instances where agents must coordinate without *full* knowledge of local or shared constraints. This encompasses problems such as multi-rover exploration, scheduling computing nodes, and swarm formation. The key structure of COSP, high degrees throughout the constraint graph, generalizes to these other problem instances. We can also capture more complex reward functions beyond satisfaction, as long as the reward

function for any $r \in R$ cannot be decomposed into a linear combination of the inputs (i.e. many agent variables are necessary inputs to the function).

4. Problem Decomposition

In this section, we discuss the decomposition approach for partitioning COSP, including a motivating example and the construction of the *Geometric Neighborhood Decomposition* (GND) heuristic. Clearly, solving smaller problems is computationally cheaper, however, decomposition schemes must be carefully constructed to preserve solution quality and address the assumptions of COSP.

4.1 A Motivating Example

We consider a simplified toy problem related to COSP to illustrate the motivation of decomposition and problem instance structures.

Definition 4.1. *The toy task assignment problem (t-COSP) consists of n agents, $a_1, .., a_n$, and n tasks, $r_1, .., r_n$. Each agent can schedule exactly one task and agent i is capable of completing any task r_j such that $j \leqslant i$. An agent only knows its own capable tasks and is unaware of what other agents can do.*

Clearly, t-COSP has exactly one solution in which agent a_i does task r_i. Similar to COSP, the maximum degree of the constraint graph is n resulting in each agent communicating with every other agent in most DCOP approaches.

Now consider the following: instead of solving one global problem, we partition the agents randomly into k sets of any size. Clearly, letting each of the k sets independently compute optimal schedules over all tasks will likely result in a sub-optimal global solution with lots of redundancies. However, if we also partition the tasks properly, we can create partitions that are both small for computational purposes and share an optimal solution with the global solution.

Proposition 1. *Let A_j be the set of agents in a random partition $1 \leqslant j \leqslant k$ as defined above and R_j be the set of tasks in partition j. Taking $R_j = \{r_i \mid a_i \in A_j\}$ results in an optimal t-COSP solution if each partition solves their sub-problem optimally.*

Proof. This proposition is trivially proved; if any agent a_i does not do task i, there will be $m + 1$ tasks for m agents to do where m is the number of agents in the partition of a_i that are indexed higher than i. □

The structure of t-COSP is similar to the structures outlined in Section 3.3 and what we observe experimentally, presented later in the paper. We desire our heuristic to partition the agents into computationally feasible sub-problems and the tasks into advantageous sub-problems for those agents to solve. Notice in t-COSP that agent a_n is the only agent capable of satisfying request r_n. Any decomposition scheme that is optimal must partition a_n and r_n into the same sub-problem. The decomposition of t-COSP is trivial, and we leverage centralized knowledge that is not available in actual problem instances of COSP, namely other agent capabilities. In real COSP instances, the number of agents capable of satisfying a request varies greatly, ranging from nearly all agents to a single agent. This information

is unknown. Therefore, ensuring agent capabilities match the prospective requests within a sub-problem is not trivial.

However, given the geometric structure of satellite scheduling, we can estimate these capabilities for groups of agents, uncovering properties of the constraint graph that can be utilized for decomposition. The underlying decomposition is motivated by the solution to t-COSP presented in Proposition 1. We create sub-problems by reasoning about estimated agent capabilities in a decentralized manner that, when solved, lead to well-founded global solutions.

4.2 Heuristic Construction

We outline the construction of the *Geometric Neighborhood Decomposition* (GND) heuristic. This heuristic originated in previous work (Zilberstein et al., 2024b). However, we present an improved version. These improvements include reasoning about temporal constraints across requests and introducing hyperparameters to construct varying degrees of non-disjoint sub-problems. The heuristic decomposes the global problem through decentralized geometric computation. The heuristic, computed individually by each agent, inherently coordinates agents without communication. This heuristic is grounded in geometry and is composed of three layers:

1. the global supply layer,
2. the inter-neighborhood delegation layer,
3. and the intra-neighborhood delegation layer.

4.2.1 Global Supply

In our application, *supply*, or the number of agents capable of satisfying a request, is important to uncover the structure of the constraint graph. The lack of knowledge of other agent variables results in agents being unaware of the global supply for a request.

Problem instances contain observations for which only a few satellites have visibility, as well as requests that the entire constellation can service. Identifying request supply enables agents to make informed decisions to minimize redundant observations and collectively service more requests.

In the traditional DCOP formulation, the supply is trivial to compute. The supply for a request, r, would be the number of agents that have a variable $x = x(s_j)$ such that $s_j = (r, h, m)$, which is known since these agents all share a constraint.

We use the geometry of the satellite orbits to estimate the visibility of a ground target for each orbital plane, obtaining an approximation of the number of satellites that could satisfy a request during a time interval. Specifically, the supply of a request, r, provided by an orbital plane, K, is determined as the duration of time that the request is in the *longitudinal cross-tracks* of the plane times the *number of agents that pass over a point per epoch*. The longitudinal cross-tracks of the orbital plane are the time intervals in which the target is within a visible range of an orbital plane. This is determined by the time interval of a request, point of closest approach, satellite slewing capabilities, and field of view. An epoch refers to an instant of time in the units in which computation is performed.

Figure 3 illustrates the geometric interval a ground target is in the cross-tracks. In the figure, Z_{ECI} denotes the rotation axis of Earth. P_1 and P_2 are the bounding planes of the cross-tracks on either side of the orbital plane. The green area then depicts the region for which any ground target might be within the visibility of a satellite in the orbital plane. The right ascension bounds depict the interval of potential visibility for a specific ground target. Using simplified geometry such as a spherical Earth and a Keplerian orbit, we can efficiently compute these time intervals via a bisection search, taking $O(\log |H|)$ time. For a request r and an orbital plane K, we denote this time interval as $I_{r,K}$.

Combining that interval with the number of agents that pass over a point per epoch, we obtain the supply estimate for an orbital plane. The latter piece of information is determined by the fixed amount of time it takes any satellite in the orbital plane to complete one full orbit of Earth, denoted o_k, and the number of agents in the plane, $|K|$. This estimation assumes that the satellites are evenly spaced in an orbital plane. However, this is not a necessary assumption. Since the geometry of the constellation is known, we can compute the supply based on the overflights that intersect the right ascension bounds even if the satellites are unevenly allocated. In these instances, an offset We compute the supply for the request r from the orbital plane K as

$$\textsc{Supply}(r, K) = I_{r,K} \cdot \frac{|K|}{o_k}.$$

The global supply can then be computed as

$$\textsc{Supply}(r) = \sum_{K \in \mathcal{K}} \textsc{Supply}(r, K).$$

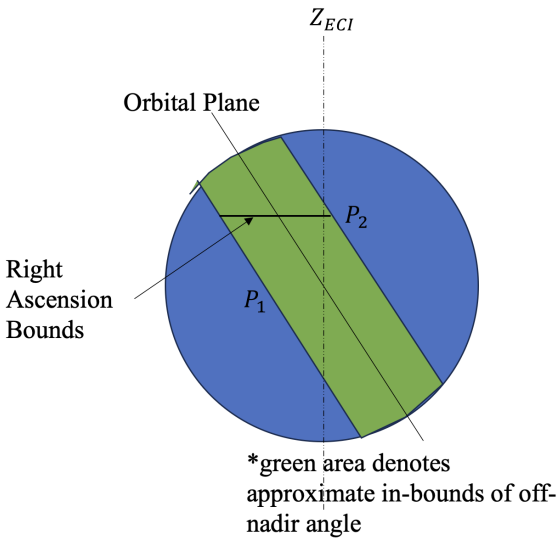

Figure 3: Estimating supply from an orbital plane. The right ascension bounds shows the temporal interval a ground target is within potential visibility of the orbital plane based on geometric constraints.

The global supply is the sum of the planar supply over all the orbital planes in the constellation. Note that this heuristic estimates supply based on geometric visibilities. However, there are other factors to consider to obtain an exact supply value. Even if a satellite has visibility of the ground target, it may still be blocked from observing due to a mandatory downlink, exact orbital mechanics and geometric constraints, or other considerations.

4.2.2 Inter-Neighborhood Delegation

The supply layer estimates groups of edges between groups of agents in the constraint graph, however, it does not perform any partitioning. The inter-neighborhood delegation layer reasons about which sub-graphs in the estimated constraint graph agents should solve.

Considering the number of satellites that can service an observation request puts the contention into perspective. Recall that the supply of many requests is nearly every satellite in the constellation. The inter-neighborhood delegation heuristic assigns a request to a fixed subset of agents, alleviating contention and reducing the problem size.

Each orbital plane, $K \in \mathcal{K}$, provides a natural grouping of agents into a large neighborhood. This is motivated by satellites in the same orbital plane experiencing similar relative geometries of Earth targets, making their view of the problem homogeneous, and the supply estimation being efficiently computable per orbital plane. Agents in the same orbital plane typically share the ability to satisfy the same requests, making the orbital planes a logical choice for large neighborhoods. We acknowledge that this neighborhood selection exploits specifics of our domain and that neighborhood selection in other domains is not always as clear-cut.

In this section and future sections, we use the term neighborhood to describe a partition of agents. It is important to note that our use of the term neighborhood is different from the typical notion of a neighborhood in DCOPs. In a DCOP, the neighborhood refers to agents that share a constraint. In our problem, agents in a neighborhood typically share many constraints, however, they also share constraints with agents in other neighborhoods. This layer of the heuristic determines which large neighborhoods (e.g. orbital planes) should prioritize servicing a request, and in turn, what other neighborhoods should de-prioritize. At an abstract level, this layer of the heuristic delegates each utility function, f_{r_j}, to fixed partitions of agents. We define the inter-neighborhood delegation heuristic based on the properties of an orbital plane and the request. The algorithm for request delegation is outlined below.

Algorithm 1 takes in the requests R, the orbital planes $\mathcal{K}$, and a hyperparameter n. The computation keeps track of which requests were delegated to which partition in the data structure *partition*, which is indexed by a plane, k. Iterating through each request, r, in order of least to most supply, we compute the heuristic value $\mathcal{H}[K]$ for each plane K. The heuristic is computed as the ratio of supply divided by the number of requests in the partition that overlap in time with r. We aim to maximize this value for each request r to create sub-problems with lower degree cliques. The sub-procedure NumberOverlapping($partition[K], r$) computes the number of requests in $partition[K]$ that temporally overlap r.

Algorithm 1 GND Inter-Neighborhood Delegation

```
procedure InterNeighborhoodDelegation(R, 𝒦, n)
    order R based on increasing Supply(r)
    partition ← ∅
    for r ∈ R do
        ℋ ← ∅
        for K ∈ 𝒦 do
            ℋ[K] ← Supply(r, K) ÷ NumberOverlapping(partition[K], r)
        end for
        𝒦_best ← arg max^n_K(ℋ)
        for K ∈ 𝒦_best do
            add r to partition[K]
        end for
    end for
    return partition
end procedure
```

The hyperparameter n denotes how many neighborhoods are selected. Note that $n = 1$ results in a complete partition of the global problem; each request will be considered by exactly one neighborhood of agents. Taking $n > 1$ results in an incomplete partition, in which n disjoint sets of agents will consider a request. For scheduling problems that are highly constrained, in that agents cannot schedule all requests and requests conflict greatly with each other, creating incomplete partitions may be advantageous. The hyperparameter n is also considered in the intra-neighborhood delegation layer of the heuristic and is discussed further in the next section.

In the arg max computation, ties are broken by selecting the plane with the minimum distance to the request target, $t(r)$, over the request window. The minimum distance is estimated using three points over the continuous interval of the request horizon, $h(r)$: the start, the end, and the midpoint. This tie-breaking can be viewed as a unique secondary estimate of the supply.

In previous work, the heuristic $\mathcal{H}$ was directly the supply, $\textsc{Supply}(r, K)$, and only $n = 1$ was considered (Zilberstein et al., 2024b). Taking into account the temporal constraint of requests and incomplete partitioning are both strategies to improve the performance and generalizability of GND across problem instances.

In Section 6.1.1 we will present the constellations, but as a consideration, we note here that $|\mathcal{K}|$ is typically small ($|\mathcal{K}| << |A|$). Therefore, while this partitioning is substantial in that we create $|\mathcal{K}|$ sets of agents each considering a subset of requests, it is not necessarily sufficient to reduce the problem scale to a desired level for DCOP algorithms. There is no bound on the distribution of agents to orbital planes, so there is no theoretical guarantee that problem instances are reduced to a desired level.

4.2.3 Intra-Neighborhood Delegation

So far, the value of the global supply heuristic and the inter-neighborhood heuristic delegate requests to a group of agents defined by the orbital plane. The final layer in the heuristic is

determining which agents in the same orbital plane should seek to schedule which requests, which enacts further partitioning. The intra-neighborhood heuristic is driven by agent biases, in which agents with the same biases form a subset of the partition.

First, we define the bias of an agent, denoted b. A bias is parameterized by a periodicity, $\rho \in \mathbb{N}$. The periodicity determines the number of unique biases and the number of agents for which the bias repeats in a neighborhood. To compute the bias, we need to index each agent in an orbital plane. For each orbital plane, K, we arbitrarily select an agent 0 and order all the agents in the plane from 0 to $|K| - 1$ by moving counter-clockwise. The bias, $b \in [0, \rho)$, is computed as an integer based on the index, i, of the agent:

$$b \equiv i \mod \rho.$$

The periodicity of the bias means all agents indexed $i + q \cdot \rho$ possess the same bias for all $q \in \mathbb{N}$. Increasing the periodicity means creating more sub-groups of agents with the same bias. The motivation for spacing agents with the same bias as opposed to selecting consecutive agents with the same bias is to diversify the sub-neighborhood's collective visibilities and geometries over time. There is no supply estimation for a request among agents within an orbital plane. In COSP, request durations tend to be sufficiently long that the majority of agents within an orbital plane are able to satisfy any request. However, this is not assumed. Spacing the bias increases the likelihood that the supply of a request in any sub-neighborhood is nonzero and is an effective strategy when uninformed. The bias for a single agent is fixed. Therefore, we assume it is known by all agents apriori with the configuration of the satellite constellation.

GND can be used with different biases in different domains. However, for COSP, we again use the geometry to define the sub-problems. We define the request target position bias which examines the latitude and longitude of the ground target, $t(r)$. For both coordinates, we add a bias if the degree times ten modular ρ is equivalent to the agent bias b. We multiply by ten to give more precision to the bias. Targets tend to be clustered in small geographic regions, so using more precision provides more diversity in this bias. We mentioned previously that many agents experience very similar relative geometry with targets in terms of time and space. The latitude and longitude bias aims to disrupt that homogeneity across agents. Figure 4 visualizes the grid pattern that occurs from this bias when projected onto latitude and longitude strips. Note that this figure is not to scale. An agent with bias $b = 1$ favors observing ground targets within the orange regions of the strips.

Let b be the bias of an agent a computed as $b \equiv i \mod \rho$ and $[\cdot]$ denote the standard integer rounding function. We define the biases as follows.

$$\textsc{LatitudeBias}(r, a) = \begin{cases} Z_1 & \text{if } \; b \equiv [lat(t(r)) \cdot 10] \mod \rho \\ Z_2 & \text{else,} \end{cases}$$

and

$$\textsc{LongitudeBias}(r, a) = \begin{cases} Z_3 & \text{if } \; b \equiv [lon(t(r)) \cdot 10] \mod \rho \\ Z_4 & \text{else.} \end{cases}$$

Each $Z_i \in \mathbb{R}$ is the heuristic value associated with the bias. The full intra-neighborhood heuristic $\zeta : R \times A \to R$ is computed as

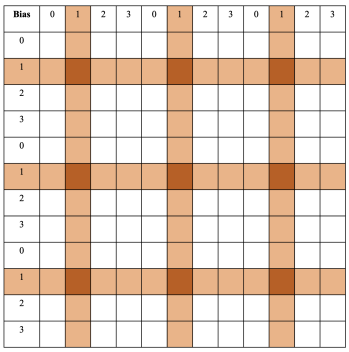

Figure 4: Latitude and longitude grid bias for an agent with $b = 1$ and $\rho = 4$.

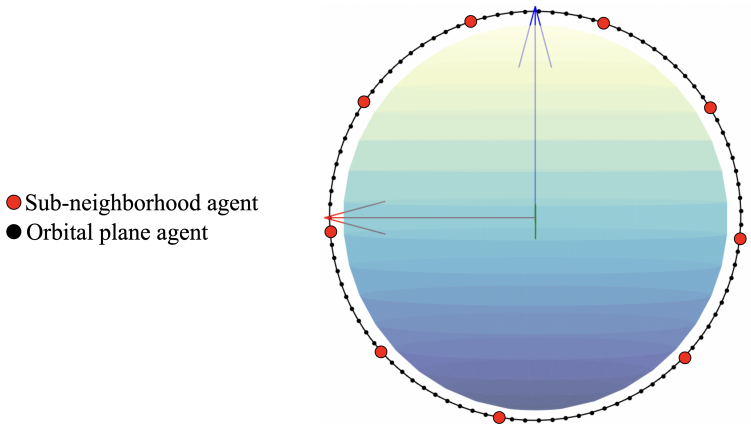

Figure 5: Partition of agents within an orbital plane.

$$\zeta(r, a) = \textsc{LatitudeBias}(r, a) + \textsc{LongitudeBias}(r, a).$$

We use ζ to divide the sub-problems within an orbital plane. We partition the agents in an orbital plane into ρ sets, in which a set is defined by agents with the same bias. Like the inter-neighborhood delegation, to partition the requests, we associate each request with n partitions of agents. We select the n biases that have the largest value of ζ. Across the intra- and inter-neighborhood heuristics, the parameter n controls how much of the state space is reachable through search. Taking $n = |\mathcal{K}| \cdot \rho$, results in the entire search space being reachable via solutions to the partitioned sub-problems. Increasing the value of ρ will increase the number of biases, which in turn affects the number of partitions of the problem. Tuning ρ enables us to create partitions of constant size, albeit at a reduction of coordination among agents. We integrate the inter- and intra-neighborhood delegation such that when delegating a request to n partitions, we delegate first to the sub-neighborhoods of agents in the best orbital plane (Section 4.2.2), before allocating to the subsequent planes. When $\rho \geqslant n$, a request will only be delegated to partitions within an orbital plane. When $\rho < n$, partitions across orbital planes can be delegated the same request.

4.2.4 Complete Heuristic

A decomposition heuristic, $\Upsilon : A \times S \rightarrow \{0, 1\}$, is a function computed by an agent that maps a request fulfillment to a Boolean value. For a single agent, a_i, the heuristic identifies the subset of requests assigned (by the heuristic) to the partition containing the agent.

We can now define GND based on the above construction. An agent, a_i, first computes the planar supply for each request. Then, an agent determines if its orbital plane is among the n orbital planes delegated to this request. If so, the agent will compute the bias value for the request and determine if it is within the agent's partition according to ζ. This procedure is the first step in coordinating sub-problems to solve with other DCOP algorithms.

When using GND to create sub-problems to deploy DCOP algorithms, an agent performs a two-step heuristic partitioning. First, an agent partitions the requests according to the inter-neighborhood delegation heuristic, and then further partitions those requests based on the intra-neighborhood delegation heuristic. Recall that a neighborhood is fixed based on the orbital plane and bias of an agent, therefore it can be computed by each agent. There are two hyperparameters of GND, ρ which determines the size of agent sub-neighborhoods, and n which controls the degree of incompleteness of request partitioning. We fix ρ based on the desired reduction in size from the global problem and refer to GND with n degrees of incompleteness as GND(n).

5. Scheduling Solutions to COSP

Before we present the algorithms, we briefly mention the procedure for generating the request fulfillments to schedule. Given the request set, R, an agent a_i determines its visibilities of the ground targets in R based on its orbit and the time windows of the requests. Using the computed visibilities, an agent greedily constructs the request fulfillments, S_{a_i}, which are the tasks that can be scheduled to satisfy requests. The point of least off-nadir angular

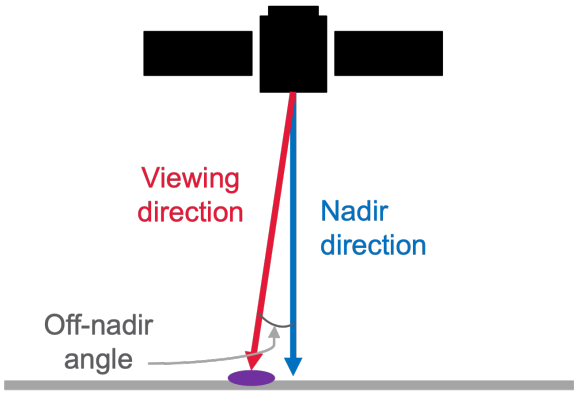

Figure 6: The off-nadir angle is calculated as the difference between the viewing direction of the target and the nadir direction. An orthogonal viewing angle typically provides optimal observation quality.

separation determines when in a visibility interval to create a request fulfillment. The interval of a request fulfillment is set to minimize the distance from the time of observation to the point of least off-nadir angular separation while ensuring that the entire fulfillment interval is within the request interval. Figure 6 illustrates the off-nadir angular separation, a metric for the proximity between the satellite and the ground target. The point of least off-nadir angular separation serves as a guide for when to observe a target for optimal viewing quality. During this task construction, request fulfillment intervals are enforced so as not to overlap with a downlink. The greedy, decentralized construction of these request fulfillments again highlights the lack of knowledge of shared constraints in COSP.

5.1 Fully Decentralized Solutions

By fully decentralized solutions, we refer to decentralized algorithms that do not rely on inter-agent communication. We present three baseline scheduling algorithms derived from a forward greedy scheduler.

5.1.1 Forward Greedy Scheduler

We implement a greedy heuristic scheduler that computes the schedule for an individual satellite without knowledge of other satellite schedules. Algorithm 2 contains the pseudocode for the decentralized scheduling procedure. This algorithm is an arbitrary greedy scheduler that functions with any heuristic over requests, denoted $\mathcal{H}_{a_i}$. For each downlink, d_j, the scheduler selects the subset of request fulfillments in which the next soonest downlink is d_j. This set of request fulfillments is denoted $S_{a_i}^{d_j}$. The request fulfillments $s_l \in S_{a_i}^{d_j}$ are then ordered based on $\mathcal{H}_{a_i}$ and iterated through, being inserted into the schedule if no constraints are violated. The scheduler performs a single pass over the ordered request fulfillments. A single forward pass scheduling approach is a benchmark for common flight capabilities (Troesch et al., 2020).

Algorithm 2 Greedy Scheduler for agent a_i

```
procedure GreedySchedule(R, S_{a_i}, D_{a_i}, C_{a_i}, H_{a_i})
    sched ← ∅
    for d_j ∈ D_{a_i} do
        compute S_{a_i}^{d_j}
        order S_{a_i}^{d_j} based on H_{a_i}
        for s_l ∈ S_{a_i}^{d_j} do
            if s_l satisfies C_{a_i} then
                addToSchedule(sched, s_l)
            end if
        end for
    end for
    return sched
end procedure
```

Algorithm 3 Random Scheduler for agent a_i

```
procedure RandomSchedule(R, S_{a_i}, D_{a_i}, C_{a_i})
    sched ← ∅
    shuffle S_{a_i}
    for s ∈ S_{a_i} do
        if s satisfies C_{a_i} then
            addToSchedule(sched, s_l)
        end if
    end for
    return sched
end procedure
```

The sub-procedure addToSchedule inserts the request fulfillment s_l into the schedule at time interval $h(s_l)$ if there is no other request fulfillment for r currently in the schedule. This is equivalent to setting $x(s_l) = 1$.

5.1.2 Baseline Scheduling Approaches

The following are the fully decentralized scheduling approaches we evaluate.

1. *Random.* Each agent $a_i \in A$ randomly shuffles its set of request fulfillments, S_{a_i}. The shuffled request fulfillments are iterated through and scheduled using the procedure addToSchedule.

2. *Greedy Start Time.* Each agent $a_i \in A$ sorts the request fulfillments, S_{a_i} based on start time of the request fulfillment. The sorted request fulfillments are iterated through and scheduled using the procedure addToSchedule. Note that this algorithm is equivalent to GreedySchedule using the start time heuristic.

3. *Greedy Portfolio.* Each agent $a_i \in A$ samples a heuristic from the portfolio of heuristics, Π, uniformly at random. The sampled heuristic defines the greedy insertion order into the schedule. Note that this algorithm is equivalent to GreedySchedule with the sampled heuristic. The portfolio consists of four heuristics: random, start time, memory usage, and off-nadir angular separation. The random heuristic assigns a random value to each request fulfillment. We note that the portfolio does not contain a dominating heuristic.

5.2 Centralized Algorithm

The centralized algorithm we employ is an adaptation of *squeaky wheel optimization* (SWO) (Joslin & Clements, 1999) shown in Algorithm 4. SWO is an incomplete centralized search algorithm. Over the course of iterations, SWO heuristically creates schedules based on priorities assigned over previous iterations. In our implementation, SWO sorts the requests based on priority, breaking ties with the least supply. It then randomly selects an available satellite to schedule the request. Requests that are not scheduled on previous iterations have their priorities increased. In subsequent iterations, requests that were not previously scheduled are attempted to be satisfied first.

The algorithm returns the best schedule found across iterations. The data structure *priorityMap* tracks the priorities of requests across iterations. A lower value equates to a higher priority. All request priorities are initialized to 0. A request scheduled in the current iteration has its priority incremented by 1, and requests that are not scheduled have their priorities reset to 0. Convergence is determined when the absolute difference in solution quality across iterations is below a threshold. In addition to the previously defined procedure addToSchedule, there are several important sub-procedures of the SWO implementation relevant to understanding an iteration of the algorithm.

- sortRequests$(R, A, S, priorityMap)$. This procedure sorts the requests in R based on their priority in *priorityMap*. A lower value equates to a higher priority. To break ties, requests are sorted in ascending order based on the number of agents in A that have a request fulfillment for the request.
- selectRequestFulfillment(r, S', A). This procedure selects a request fulfillment from S', the set of all feasible request fulfillments across all agents to satisfy the request r. It returns a tuple (s, a) of the selected request fulfillment and the scheduling agent. If no feasible request fulfillments exist for the request, the tuple $(\varnothing, \varnothing)$ is returned.

Algorithm 4 Squeaky Wheel Optimization

```
procedure SqueakyWheelSchedule(R, S, D, C, A, maxIterations)
    bestSchedule ← ∅
    priorityMap[r] ← 0,  r ∈ R
    i ← 0
    while i < maxIterations ∧ not converged do
        sortRequests(R, A, S, priorityMap)
        currentSchedule ← ∅
        S' ← copy(S)
        for r ∈ R do
            s, a ← selectRequestFulfillment(r, S', A)
            if s = ∅ then
                priorityMap[r] ← 0
            else
                addToSchedule(currentSchedule[a], s)
                removeInfeasibleRequestFulfillments(S', s, a)
                priorityMap[r] ← priorityMap[r] + 1
            end if
        end for
        if currentSchedule is better than bestSchedule then
            bestSchedule ← currentSchedule
        end if
        i ← i + 1
    end while
    return bestSchedule
end procedure
```

Several selection heuristics were evaluated. However, random selection provided the best balance of execution time and solution quality. Squeaky wheel optimization can thrash over many iterations when evaluating the problem in a similar fashion via unchanging priorities and deterministic heuristics. Using a random selection heuristic diversifies the traces of the scheduler across iterations.

- removeInfeasibleRequestFulfillments(S', s, a). This procedure removes from S' the request fulfillments that are no longer feasible after scheduling s for agent a. Note that only request fulfillments for agent a that exist within the same downlink window could become infeasible. Let d_j be the next downlink for agent a after $h(s)$. A request fulfillment, s_j is removed if setting $x(s_j) = 1$ violates c_{d_j} or $c_{s_j,s}$. This procedure ensures that the request fulfillments in S' can be scheduled without violating constraints. In addition, all request fulfillments for $r(s)$ are removed from S' for efficiency. However, note that no request fulfillments that satisfy r will be scheduled in an iteration after request r is considered during the inner loop.

Before an iteration, the set of request fulfillments, S, is copied into the data structure S' to enable the removal of request fulfillments that become infeasible. An iteration begins by sorting the requests in the sub-procedure sortRequests, which is detailed above. The sorted requests are then iterated through and a request fulfillment to satisfy each request is selected in the procedure selectRequestFulfillment. The *priorityMap* is then updated depending on whether or not a request fulfillment, s, was found to satisfy r. If there is a request fulfillment to satisfy r, it is scheduled in the procedure addToSchedule, and S' is updated by removing the infeasible request fulfillments as a result of scheduling s.

5.3 Neighborhood Stochastic Search

The *Neighborhood Stochastic Search*(NSS) algorithm is a DCOP algorithm developed for COSP. It was first presented in prior work (Zilberstein et al., 2024b) and extends the request satisfaction variation of BD (Parjan & Chien, 2023) to more precisely fit both COSP and DCOPs, scale to large problem instances, and enable scheduling with resource constraints. We present the pseudo-code in Algorithm 5 and mention key sub-procedures. The first step for an agent is to compute, using a decomposition heuristic, Υ, the sub-problem the agent is involved in solving. We denote this sub-problem as $\mathcal{N}$, which is itself a DCOP consisting of agents, $A_{\mathcal{N}} \subseteq A$, and requests, $R_{\mathcal{N}} \subseteq R$.

- computeSubProblem$(a_i, A, R, S_{a_i}, \Upsilon)$. This procedure returns the neighborhood of agent a_i and the requests for the neighborhood to schedule using the decomposition heuristic Υ. Each agent knows the request set, R, the other agents, A, and the orbital planes and fixed bias. The neighborhood of agents is the subset of agents of A such that the orbital plane and bias are the same as the computing agent. A subset of the request set is subsequently selected according to the decomposition heuristic as requests for which the value of $\Upsilon(a_i, s) > 0$ for some $s \in S_{a_i}$ such that $r(s) = r$. The neighborhood defines the agents that an agent will communicate with. Note that this computation produces neighborhoods that are reflexive and transitive.

Algorithm 5 Neighborhood Stochastic Search for agent a_i

```
procedure NSS(R, S_{a_i}, D_{a_i}, A, ϒ, maxIterations)
    N = computeSubProblem(a_i, A, R, S_{a_i}, ϒ)
    sched = initialSolution(N, S_{a_i}, D_{a_i}, C_{a_i})
    while i < maxIterations ∧ not converged do
        com = message(A_N, sched)
        shuffle R_N
        for r ∈ R_N do
            assigned = stochasticUpdate(r, sched, com)
            if assigned = true then
                scheduled = schedule(r, sched, S_{a_i})
            end if
        end for
        i ← i + 1
    end while
    return sched
end procedure
```

- initialSolution($\mathcal{N}, S_{a_i}, D_{a_i}, C_{a_i}$). This procedure constructs an initial schedule for agent a_i. We use random initialization, in which agents construct initial schedules using Algorithm 3. This initialization scheme is typical for DSA variants (Zhang et al., 2005).

- message($A_{\mathcal{N}}$, sched). This procedure encapsulates the message exchange between agents in a neighborhood. Agent a_i messages the subset of $R_{\mathcal{N}}$ that it satisfied in the previous iteration to each agent in $A_{\mathcal{N}}$. In addition, agent a_i receives the broadcast from each agent in its neighborhood and compiles the received information in the data structure *com*.

- stochasticUpdate(r, sched, com). This procedure updates the assignment of the agent to the request based on the communicated information. By assignment, we refer to whether or not the request *should* be scheduled by this agent. Let m be the number of agents that satisfied request r according to the broadcast, and P_u be a probability to be discussed later. The assignment of a request r is stochastically updated in the following ways.

  - If agent a_i is not assigned to r and $m = 0$, a_i assigns to r.
  - If agent a_i is not assigned to r and $m > 0$, a_i remains unassigned to r.
  - If agent a_i is assigned to r and $m = 0$, a_i will unassign with probability P_u.
  - If agent a_i is assigned to r and $m > 0$, a_i will unassign with probability $\frac{m-1}{m}$.

- schedule($r, sched, S_{a_i}$). This procedure attempts to schedule a request fulfillment for request r in the current schedule. If an assigned request fulfillment satisfies C_{a_i} it is immediately inserted into the schedule. Otherwise, the scheduler can remove a conflicting request fulfillment from the schedule to free up resources. The removed

request fulfillments are selected as the closest start time to the request fulfillment to insert. Agent a_i remains assigned to removed request fulfillments and may attempt to re-schedule these in subsequent iterations based on the stochastic assignments. Allowing agents to de-schedule requests enables the algorithm to overcome getting stuck at local maxima. Note that we consider all requests of equal priority and there is no temporal flexibility in the start or end times of request fulfillments. The scheduler is amenable to these extensions and in future work, we can include continuous variables in COSP to model temporal flexibility.

We evaluate several variations of the NSS algorithm with different heuristics and partitioning. NSS-Random serves as a baseline of GND and refers to NSS with a random decomposition heuristic. NSS-Random randomly creates partitions of agents and requests. With no other knowledge of the constraint graph, doing more sophisticated partitioning is challenging, therefore NSS-Random serves as a suitable, naive baseline for GND. NSS-GND(n), refers to NSS with the GND(n) heuristic.

The stochastic search performed in NSS mimics the search of BD with some key distinctions: decomposition to reduce size, de-allocating to overcome local minima, and scheduling with resource constraints. Note that NSS relies on the parameter P_u. We use $P_u = 0.7$ as published by the authors of the BD algorithm (Parjan & Chien, 2023), but mention that changing P_u, had minimal effect on the performance of NSS.

5.4 Theoretical Analysis of Algorithms

We begin by examining the complexity of NSS which we present in Propositions 2 and 3. We define $A_{\mathcal{N}}$ as the largest set of agents in a sub-problem and $R_{\mathcal{N}}$ as the largest set of requests in a sub-problem. We also define L, the maximum size of a satellite's schedule to more exactly capture the complexity. The value of L is driven by the horizon, H, the requests, R, and an agent's constraints, C_{a_i}. In practice $L << |R|$ (and $|A_{\mathcal{N}}|$, $|R_{\mathcal{N}}|$). Checking if a request fulfillment satisfies C_{a_i} and inserting into a schedule are both $O(\log L)$ operations. We omit L and other factors in the complexity of NSS that are subsumed by larger factors.

Proposition 2. *In every iteration, NSS requires an agent to send $O(|A_{\mathcal{N}}|)$ number of messages each of size $O(|R_{\mathcal{N}}|)$.*

Proof. The only message exchange occurs in the procedure Message in which an agent sends the subset of its sub-problem's requests that it satisfied to all agents in the sub-problem, which are at most $|R_{\mathcal{N}}|$ and $|A_{\mathcal{N}}|$ respectively. □

Proposition 3. *In every iteration, NSS requires an agent to perform $O(|A_{\mathcal{N}}|\cdot|R_{\mathcal{N}}|)$ operations.*

Proof. To compute the probabilities used in StochasticUpdate, an agent must iterate through the communication received from the Message procedure, which is $O(|A_{\mathcal{N}}|\cdot|R_{\mathcal{N}}|)$. The main loop of NSS iterates through at most $O(|R_{\mathcal{N}}|)$ requests, performing $O(\log L)$ computation for each. □

Algorithm	Computation	Communication
Random	$\|R\|\cdot\log L$	N/A
Greedy Start Time	$\|R\|\cdot\log\|R\|$	N/A
Portfolio Greedy	$\|R\|\cdot\log\|R\|$	N/A
NSS	$k\cdot\|R_{\mathcal{N}}\|\cdot\|A_{\mathcal{N}}\|$	$k\cdot\|R_{\mathcal{N}}\|\cdot\|A_{\mathcal{N}}\|$
BD, DSA, MGM	$k\cdot\|R\|\cdot\|A\|$	$k\cdot\|R\|\cdot\|A\|$

Table 3: $O(\cdot)$ complexity of decentralized algorithms.

We summarize the per-agent computational and communication complexity of the decentralized algorithms in Table 3. It is assumed that all agents have knowledge of the requests. Therefore, the fully decentralized algorithms incur no communication.

The centralized algorithm, SWO, has a computational complexity of $O[k\cdot(|R|^2+|R|\cdot|A|)]$. This cost is incurred by the centralized node which has arbitrary computing power. Centralized algorithms require sending the final schedules to each agent, resulting in an $O(L\cdot|A|)$ communication cost. The NSS algorithm incurs a communication cost proportional to the size of the sub-problem. For NSS, SWO, and other relevant algorithms we define k, the number of iterations.

The complexity shows that the fully decentralized algorithms are the most efficient. In comparison to MGM, DSA, or BD, NSS achieves a complexity parameterized by $|A_{\mathcal{N}}|$ and $|R_{\mathcal{N}}|$ in each iteration as opposed to $|A|$ and $|R|$.

6. Experimental Results

We select hyperparameters of GND such that the global problem is reduced in size by an order of magnitude. We set ρ such that the largest neighborhood of satellites is at least 10 times smaller than the total constellation size. In the intra-neighborhood delegation heuristic, ζ, we set Z_i such that the resultant ordering is based on latitude, then longitude. An example assignment is $Z_1 = 2$, $Z_3 = 1$, and $Z_2 = Z_4 = 0$. We then evaluate GND(n) with $n = 1, 2, 4$ and 8. For both NSS and SWO, we set $maxIterations = 20$ and determine convergence using a solution difference threshold of 0.001% satisfaction.

6.1 Experimental Setup

6.1.1 Satellite Constellations

We simulate two constellations inspired by operational low Earth orbit constellations (Planet, 2023). The first constellation is modeled on the Dove constellation from Planet, and we refer to it as Planet. There are 200 agents in total divided across 4 orbital planes at 600 km altitude. The constellation has two near sun-synchronous orbital planes at 95° inclinations composed of 95 satellites each. There are an additional two orbital planes at 52° inclinations with 5 satellites each. Each satellite has a single sensor that can slew to 60° off of nadir and an on-board memory capacity of 125 GB.

The second constellation is derived from a Walker constellation, and based on the Skysat constellation from Planet. We refer to this constellation as Walker. This constellation has 6 orbital planes with 14 satellites each at an 88° inclination. There is an overlay of 2 orbital

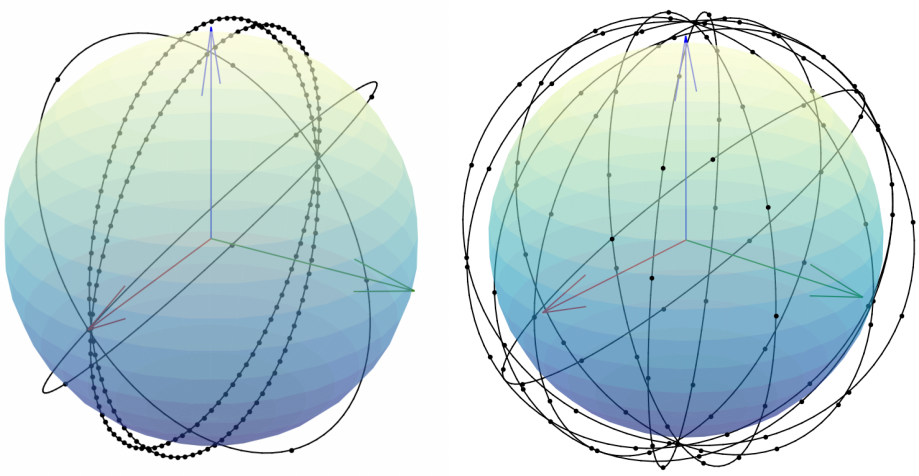

Figure 7: Visualization of the satellite constellations: Planet-inspired (left) and Walker constellation with overlay (right). The dots represent a single satellite along the path defined by the orbital plane.

planes at a 51.6° inclination with 12 satellites. This overlay provides more coverage at low declinations and is in a similar orbit as the International Space Station. Each satellite has a single sensor that can slew to 45° off of nadir and an on-board memory capacity of 125 GB. Figure 7 shows the two constellations centered around a sphere.

6.1.2 Downlinks

We consider two ground stations: the ASF Near Space Network Satellite Tracking Ground Station and the Guam Remote Ground Terminal System. A downlink is modeled as a constant bit stream of 62.5 MB/s for the duration of visibility of the ground stations.

6.1.3 Observation Request Campaigns

The target set, T, is composed of 634 globally distributed ground targets (cities and volcanoes). Figure 8 illustrates the distribution of targets on the surface of Earth.

We generate a campaign by selecting a random subset of targets from T that has a size of at least 75% of T. For large problem instances, a random periodicity is selected in the range $[4, 12]$. A periodicity of p means the constellation is requested to observe each target once within p evenly spaced intervals during the scheduling horizon, resulting in p requests for a target. For small problem instances, we reduce the periodicity and randomly remove requests to obtain a problem instance with a smaller set of requests. The start of the scheduling horizon is randomly initialized. The end of the horizon is fixed at 24 hours after the selected start time. The amount of memory required by a request fulfillment is sampled from a normal distribution with mean 50 MB and standard deviation 10 MB. The interval of time required to schedule a request fulfillment is fixed at 63 seconds (3 seconds for the observation and 30 seconds on either side for slewing and processing).

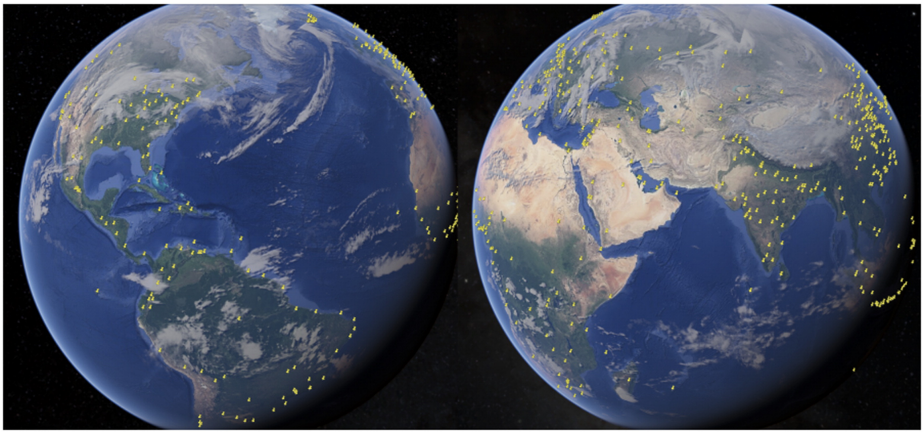

Figure 8: Visualization of the set of point targets on Earth, T.

Generating campaigns in this manner produces hard problem instances. By hard, we refer to the constraint graph structure mentioned in Section 3. Despite fixing the scheduling horizon at one day, it is the density of requests during the window (i.e. requests per epoch) that drives the difficulty of the scheduling problem.

6.2 Evaluation

6.2.1 Geometric Neighborhood Decomposition

We begin by examining large problem instances and the decomposition of GND on these instances. Figure 9 plots the distribution of request supply for the global problem and different variations of GND(n). The distribution of the global problem matches what we describe for t-COSP and elsewhere in Section 3; $\Omega(|R|)$ cliques of size $\Omega(|A|)$ as well as cliques of constant size, composed of just a few satellites. For the Planet constellation consisting of 200 agents, the first and third quantiles are 5 and 141 respectively. For the Walker constellation, which has 108 agents, these values are 15 and 29, however, the spread of outliers is much larger.

The partition sizes of GND(n) variations match what we expect; GND(1) creates the smallest sub-problems, shrinking the supply distribution by an order of magnitude. As n increases, so does the supply for individual requests. Interestingly, we see minimal supply change between GND(4) and GND(8) for both Planet and Walker constellations, suggesting that the majority of supply for a request is contained within 4 neighborhoods.

We also examine the error in the GND supply estimation as shown in Figure 10. The plot shows the average normalized error of the estimation. This error is computed as the absolute difference between estimated supply and actual supply, normalized based on the size of the constellation (e.g. number of agents). The supply estimation is computed as described in Section 4.2.1 while the actual supply for a request, r, is computed as the exact number of agents that have a task $s_j = (r, h, m)$. This equivalently describes the relative error in the degree estimation of a node in the constraint graph. We find that the Walker constellation has a much lower error in the supply estimation, averaging around a 2% error in the size of the constellation that reduces slowly and linearly as the problem size increases.

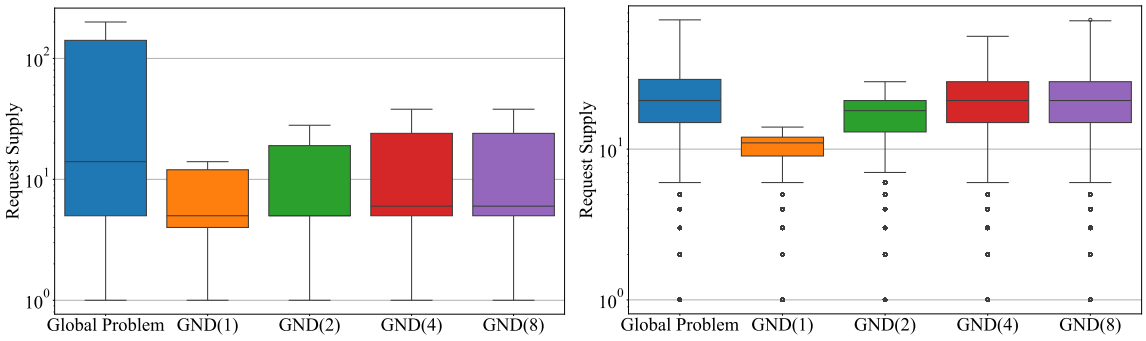

Figure 9: Distribution of request supply across 100 problem instances after the different decomposition schemes for Planet constellation (left) and Walker constellation (right).

In contrast, GND has a larger absolute error for the Planet constellation, peaking at 6.5%, and decreases much faster. This is partially due to the size of the Planet constellation being nearly double that of the Walker constellation, therefore the average degree of nodes is larger, as seen in Figure 9.

GND utilizes estimates that are employed in the decomposition. However, we see that this computation is not perfect, and includes some error. The error is relatively low compared to the global problem size and we confirm that this error has no degradation on the scheduling performance by evaluating GND using the exact supply value. We show in the next sections that NSS-GND can provide efficient, high-quality scheduling solutions.

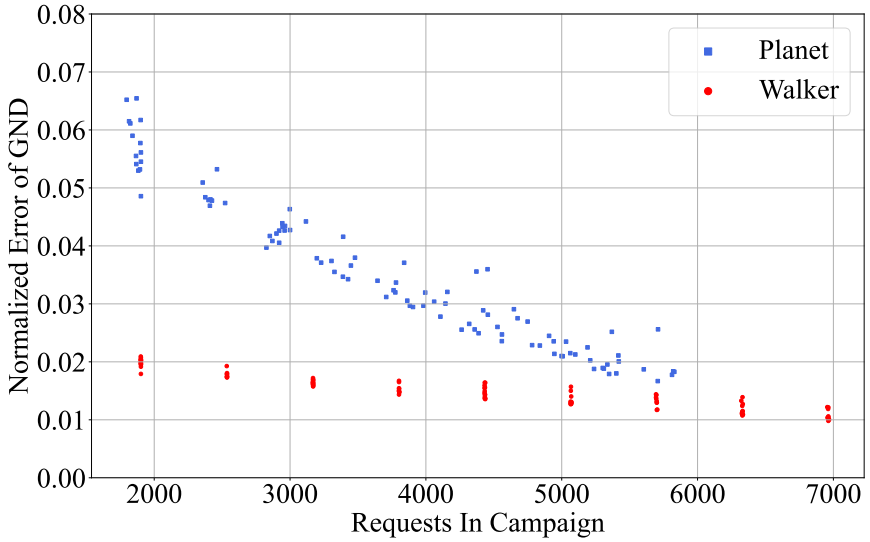

Figure 10: Average normalized error of GND supply estimation across 100 large problem instances.

6.2.2 Small Problem Instances

To demonstrate the effectiveness of our algorithms, we compare their performance on small problem instances to an optimal solution. By small problem instances, we refer to a small request set, which we upper bound at 500 requests. We use a branch and bound search to obtain an optimal schedule for each satellite in the constellation to maximize the total number of satisfied requests. The branch and bound algorithm can only execute on small problem instances due to computational constraints and memory limitations. We report the average gap in satisfaction percentage to the optimal solution of the algorithms over 50 randomly generated small problem instances with between 400 and 500 requests in Tables 4 and 5.

It is important to note that small problem instances are typically under-constrained, especially for the Walker constellation, which has relatively uniform coverage of Earth. This is illustrated by the high performance of the Random algorithm. However, as the problem instances scale in the number of requests, the constraints on the system increase, and so does the necessity for coordination.

The results show that the centralized solution, SWO, achieves near-optimal performance. The NSS algorithms also achieve close to optimal performance, while the fully decentralized

Algorithm	**Opt. Gap (%)**	**Time (ms)**	**Messages (KB)**
Random	3.856	< 1	0
Greedy Start Time	5.152	< 1	0
Portfolio Gredy	4.544	< 1	0
NSS-Random	3.187	3.375	1, 4303.712
NSS-GND(1)	3.072	< 1	106.718
NSS-GND(2)	1.713	< 1	181.392
NSS-GND(4)	1.784	< 1	207.886
NSS-GND(8)	1.784	< 1	207.886
BD	2.160	10.775	11, 895.264
SWO	0.020	644.925	0.489
B&B	0.0	373, 174.350	0.489

Table 4: Results of algorithms on 50 small problems for the Planet constellation.

Algorithm	**Opt. Gap (%)**	**Time (ms)**	**Messages (KB)**
Random	0.188	< 1	0
Greedy Start Time	3.980	< 1	0
Portfolio Gredy	0.284	< 1	0
NSS-Random	0.632	3.02	604.730
NSS-GND(1)	1.192	< 1	122.336
NSS-GND(2)	0.308	< 1	234.564
NSS-GND(4)	0.236	1.08	385.093
NSS-GND(8)	0.272	1.46	411.798
BD	0.408	4.18	2, 451.242
SWO	0.0	305.98	0.499
B&B	0.0	1, 129.0	0.499

Table 5: Results of algorithms on 50 small problems for the Walker constellation.

solutions are significantly further from optimal for the Planet constellation. In comparison to BD, the NSS-GND algorithms achieve mostly higher request satisfaction while possessing faster run times and procuring an order of magnitude fewer messages. Notably, most variations of NSS-GND outperform NSS-Random in both solution quality and efficiency. These results support the efficacy of GND in generating advantageous sub-problems and the theoretical analysis of NSS.

6.2.3 Large Problem Instances

We evaluate each scheduling algorithm against 100 randomly generated large problem instances. Note that solving a large problem instance optimally would likely take longer than the age of the universe.

Figure 11 shows the reward achieved by each algorithm as a function of problem size. The NSS-GND algorithms outperform the fully decentralized solutions and the NSS-Random variation. The results enforce that GND creates effective sub-problems relative to random decomposition. In comparison to the centralized solution, there remains a gap between any of the decentralized approaches. There is also a 3% gap in satisfaction between the best NSS variation and the BD algorithm. However, we will see that what NSS trades off in solution quality, it gains in crucial metrics for distributed deployment: message volume and run time.

Figure 11 also shows that as the density of requests grows, problem instances become more difficult. The constellation as a whole cannot satisfy all the requests, therefore coordinating to reduce redundancy of observations becomes more and more crucial.

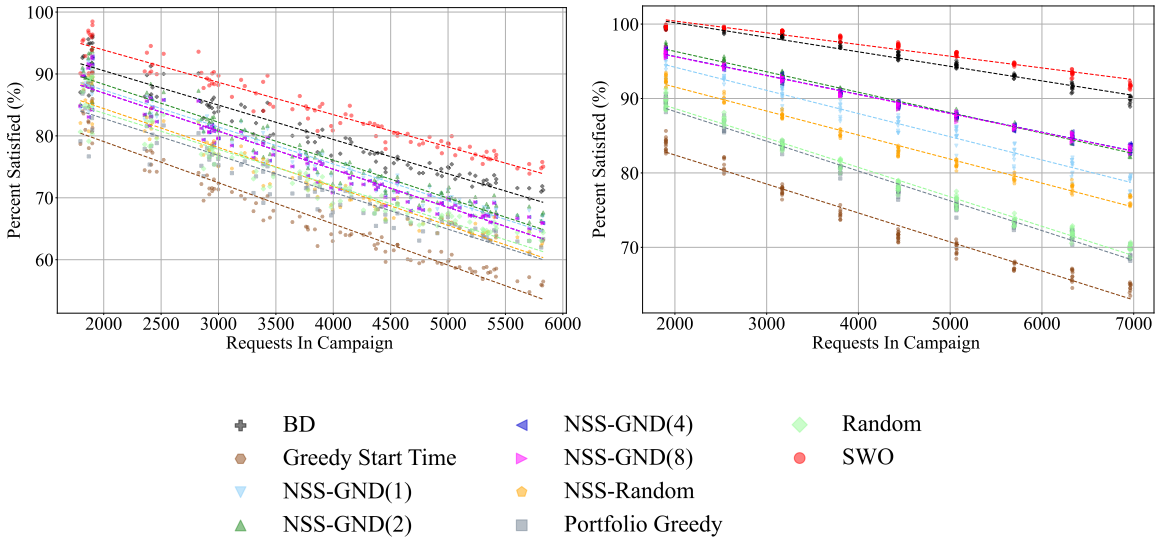

Figure 11: Percent of satisfied requests achieved by each algorithm across 100 large problem instances presented as a scatter plot for Planet constellation (left) and Walker constellation (right). Each point represents the results of a single problem instance and we regress a line of best fit for each algorithm.

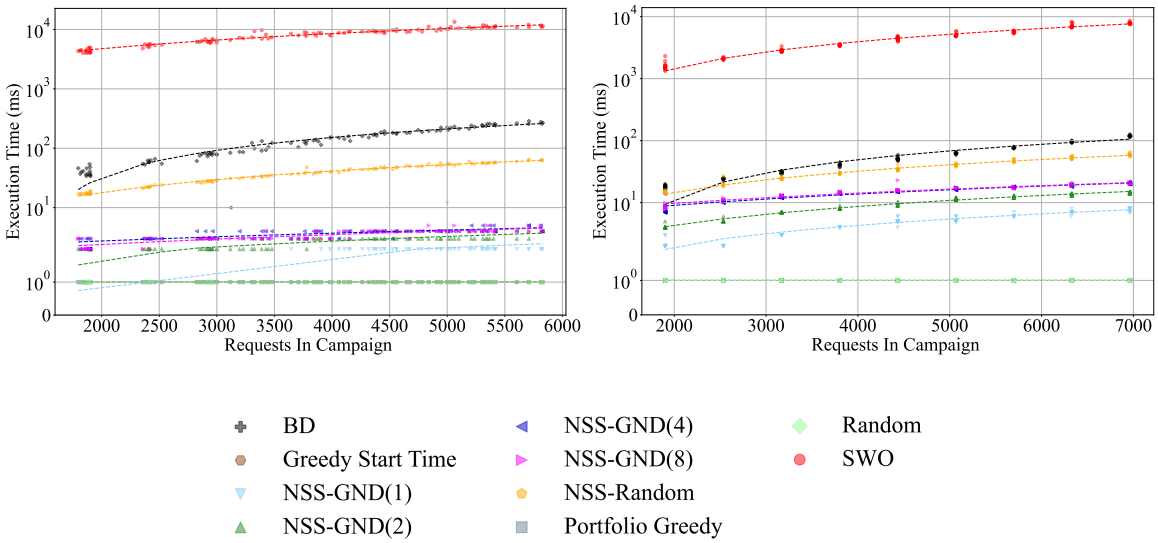

Figure 12: Execution time of each algorithm across 100 large problem instances presented as a scatter plot for Planet constellation (left) and Walker constellation (right). Each point represents the results of a single problem instance and we regress a line of best fit for each algorithm.

Figure 12 depicts the make-span execution time of each algorithm as a function of the problem size. Simulations are executed in Java. Notice the non-linearity of the y-axis. The execution times of the decentralized approaches are reported as the max run time over all agents. The NSS algorithms are an order of magnitude slower than the fully decentralized approaches. Critically, NSS-GND algorithms achieve run times that are nearly 2 orders of magnitude faster than BD and one order of magnitude faster than NSS-Random. The centralized approach is another two orders of magnitude slower than the BD algorithm. These results support the theoretical analysis from Section 5.4.

Finally, we see in Figure 13 the total message volume of the varying approaches reported in bytes. Notice again the non-linearity of the y-axis. We see that the NSS-GND algorithms achieve a total message volume that is also around 2 orders of magnitude less than BD and slightly less than NSS-Random.

While centralized algorithms will nearly always provide higher-quality solutions, our GND-based solution is highly effective in the decentralized context, outperforming all the decentralized baselines. This demonstrates that high-quality schedules can be produced in very large-scale constellations by utilizing problem decomposition. Given the unique challenges of deploying autonomy to satellites, we desire solutions that are lightweight and scalable. The BD algorithm, while effective at producing high-quality solutions, incurs costs that would not be feasible for deployment. We have shown that through informed partitioning, we can trade off some solution quality for huge reductions in execution time and message volume. We have also shown that, relative to a naive partitioning approach, GND is effective given the assumptions of COSP. NSS-GND is an improvement of more expensive

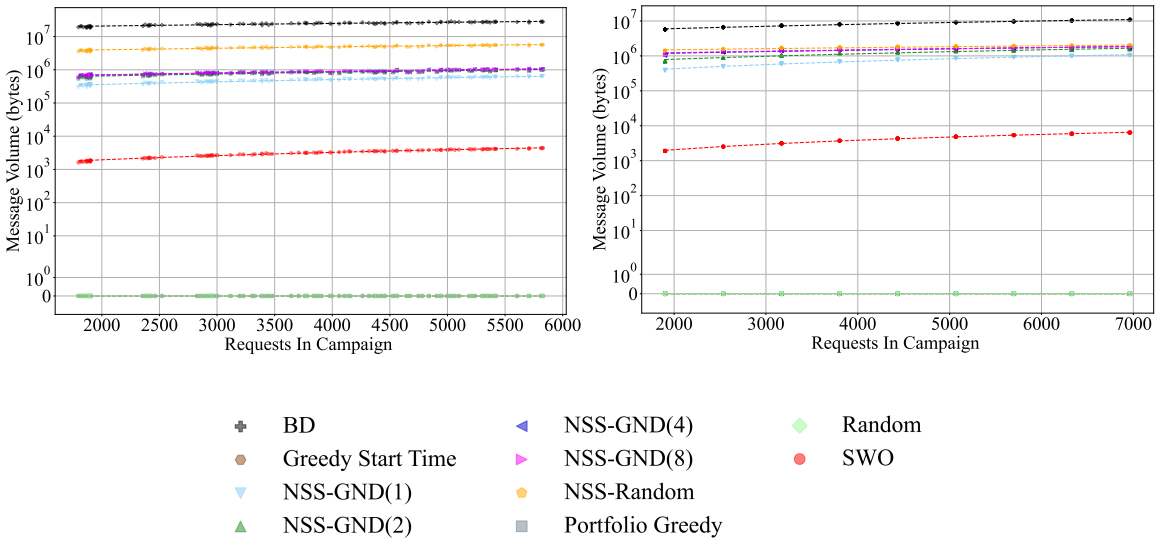

Figure 13: Message volume of each algorithm across 100 large problem instances presented as a scatter plot for Planet constellation (left) and Walker constellation (right). Each point represents the results of a single problem instance and we regress a line of best fit for each algorithm.

algorithms in terms of computation and communication and would be much more suitable for deployment to a multi-satellite constellation.

7. Conclusion

A major barrier to applying existing DCOP algorithms to large-scale, real-world problems is their computation and communication complexities, specifically when dealing with highly connected constraint graphs. We propose the *Neighborhood Stochastic Search* (NSS) algorithm, a decomposition-based approach to the multi-satellite constellation observation scheduling problem (COSP) that is efficient in both time and message complexity and can scale to problems orders of magnitudes larger than previous approaches. While not offering quality guarantees, we show that solving well-constructed sub-problems can generate high-quality global solutions while reducing the overall costs burdened by each agent. The *Geometric Neighborhood Decomposition* (GND) heuristic exploits the geometric nature of COSP, enabling decomposition in the face of limited agent knowledge. The assumptions of COSP and elements of GND extend to other domains, especially those in which computation and communication may be limited, geometry or topology play a key role, and rewards are dictated by unknown agent information. These factors are often overlooked in applications of distributed optimization, but are crucial to real-world deployment. Examples of other relevant applications include multi-agent path finding (Salzman & Stern, 2020), mobile sensor teams (Pertzovsky, Zivan, & Agmon, 2024), and UAV coordination (Pujol-Gonzalez et al., 2013). Future work would adapt NSS to these other domains with variations of GND, such as ones that perform dynamic neighborhood formation, use exact

supply computation or relevant estimation, and leverage domain-specific biases. The general framework of NSS with a decomposition heuristic applies to a broad range of distributed optimization problems.

As DCOPs are leveraged for more and more applications, extensions of the DCOP model have arisen. These include *dynamic DCOPs* (Macarthur et al., 2011; Yeoh et al., 2015; Hoang et al., 2022), *continuous DCOPs* (Liao & Hoang, 2024; Nguyen & Yao, 2012; Hoang et al., 2020), *privacy aware DCOPs* (Grinshpoun et al., 2019; Grinshpoun & Tassa, 2016; Tassa et al., 2017), *communication aware DCOPs* (Rachmut, Zivan, & Yeoh, 2022), and more. Exploring COSP in these models is relevant to future work. COSP can be dynamic in which active agents and requests change over time. Instead of fixing the interval of a task, we can reason about the physical dynamics of maneuvering a satellite's instrument in a continuous setting. Federating multiple constellations of satellites operated by different organizations into one observing system would require agents to communicate securely and maintain privacy. Modeling interference and imperfect communication is relevant to satellite cross-links, and communication constraints can be bottlenecks of distributed deployment.

Beyond the application of scheduling a satellite constellation, many large multi-agent systems come with limiting constraints, and developing algorithms that work within those constraints is essential. Current DCOP work suffers from a lack of scalability to systems that possess computation and communication limitations. Partitioning the global problem is one strategy to enable the broader application of DCOP solutions that have varying complexity. In addition, an underlying assumption of many DCOP solutions, that agents know shared constraints, does not apply to every domain. Solutions that conform to this reality are a necessity for the application of DCOP solvers to real-world domains.

Acknowledgments

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004). Government sponsorship acknowledged. We also thank the anonymous reviewers for their valuable feedback.

References

Augenstein, S., Estanislao, A., Guere, E., & Blaes, S. (2016). Optimal scheduling of a constellation of Earth-imaging satellites, for maximal data throughput and efficient human management. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 26, pp. 345–352.

Boerkoel, J., Mason, J., Wang, D., Chien, S., & Maillard, A. (2021). An efficient approach for scheduling imaging tasks across a fleet of satellites. In *Proceedings of the International Workshop on Planning and Scheduling for Space*.

Bonnet, G., & Tessier, C. (2007). Collaboration among a satellite swarm. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 1–8.

Bonnet, G., & Tessier, C. (2008). Coordination despite constrained communications: A satellite constellation case. In *Proceedings of the National Conference on Control Architectures of Robots*, pp. 89–100.

Chatterjee, A., & Tharmarasa, R. (2024). Multi-stage optimization framework of satellite scheduling for large areas of interest. *Advances in Space Research*, *73*(3).

Chien, S., Candela, A., Zilberstein, I., Rijlaarsdam, D., Hendrix, T., & Dunne, A. (2024). Leveraging commerical assets, edge computing, and near real-time communications for an enhanced New Observing Strategies (NOS) flight demonstration. In *Proceedings of the IEEE Geoscience and Remote Sensing Symposium*.

Chien, S., Sherwood, R., Tran, D., Cichy, B., Rabideau, G., Castano, R., Davis, A., Mandl, D., Frye, S., Trout, B., et al. (2005). Using autonomy flight software to improve science return on Earth Observing One. *Journal of Aerospace Computing, Information, and Communication*, *2*(4), 196–216.

Clement, B. J., Durfee, E. H., & Barrett, A. C. (2007). Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research*, *28*, 453–515.

Cohen, L., Galiki, R., & Zivan, R. (2020). Governing convergence of Max-sum on DCOPs through damping and splitting. *Artificial Intelligence*, *279*, 103212.

Deng, Y., & An, B. (2020). Speeding up incomplete gdl-based algorithms for multi-agent optimization with dense local utilities. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 31–38.

Dorri, A., Kanhere, S. S., & Jurdak, R. (2018). Multi-agent systems: A survey. *IEEE Access*, *6*, 28573–28593.

Dunkel, E. R., Swope, J., Candela, A., West, L., Chien, S. A., Towfic, Z., Buckley, L., Romero-Cañas, J., Espinosa-Aranda, J. L., Hervas-Martin, E., et al. (2023). Benchmarking deep learning models on myriad and snapdragon processors for space applications. *Journal of Aerospace Information Systems*, *20*(10), 660–674.

Eddy, D., & Kochenderfer, M. J. (2021). A maximum independent set method for scheduling Earth-observing satellite constellations. *Journal of Spacecraft and Rockets*, *58*(5), 1416–1429.

Fioretto, F., Pontelli, E., & Yeoh, W. (2018). Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research*, *61*, 623–698.

Fioretto, F., Yeoh, W., & Pontelli, E. (2016). Multi-variable agents decomposition for DCOPs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.

Gershman, A., Meisels, A., & Zivan, R. (2009). Asynchronous forward bounding for distributed COPs. *Journal of Artificial Intelligence Research*, *34*, 61–88.

Globus, A., Crawford, J., Lohn, J., & Pryor, A. (2004). A comparison of techniques for scheduling Earth-observing satellites. In *Proceedings of the Conference on Innovative Applications of Artificial Intelligence*.

Grinshpoun, T., & Tassa, T. (2016). P-SyncBB: A privacy preserving branch and bound DCOP algorithm. *Journal of Artificial Intelligence Research*, *57*, 621–660.

Grinshpoun, T., Tassa, T., Levit, V., & Zivan, R. (2019). Privacy preserving region optimal algorithms for symmetric and asymmetric DCOPs. *Artificial Intelligence*, *266*, 27–50.

He, L., Liu, X., Laporte, G., Chen, Y., & Chen, Y. (2018). An improved adaptive large neighborhood search algorithm for multiple agile satellites scheduling. *Computers & Operations Research*, *100*, 12–25.

Hirayama, K., & Yokoo, M. (1997). Distributed partial constraint satisfaction problem. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, pp. 222–236.

Hoang, K. D., Fioretto, F., Hou, P., Yeoh, W., Yokoo, M., & Zivan, R. (2022). Proactive dynamic distributed constraint optimization problems. *Journal of Artificial Intelligence Research*, *74*, 179–225.

Hoang, K. D., Fioretto, F., Yeoh, W., Pontelli, E., & Zivan, R. (2018). A large neighboring search schema for multi-agent optimization. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, pp. 688–706.

Hoang, K. D., Yeoh, W., Yokoo, M., & Rabinovich, Z. (2020). New algorithms for continuous distributed constraint optimization problems. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 502–510.

Joslin, D. E., & Clements, D. P. (1999). Squeaky wheel optimization. *Journal of Artificial Intelligence Research*, *10*, 353–373.

Kangaslahti, A., Mason, J., Swope, J., Holzmann, T., Davies, A. G., Chien, S., Harrison, T., & Walter, J. J. (2024). Sensorweb systems for global high-resolution monitoring of environmental phenomena. *Journal of Aerospace Information Systems*, *21*(8), 616–627.

Khan, M., Tran-Thanh, L., Ramchurn, S., & Jennings, N. (2018a). Speeding up GDL-based message passing algorithms for large-scale DCOPs. *Computer Journal*, *61*, 1639–1666.

Khan, M., Tran-Thanh, L., Yeoh, W., & Jennings, N. (2018b). A near-optimal node-to-agent mapping heuristic for GDL-based DCOP algorithms in multi-agent systems. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 1604–1612.

Liao, X., & Hoang, K. D. (2024). A population-based search approach to solve continuous distributed constraint optimization problems. *Applied Sciences*, *14*(3), 1290.

Lin, X., Chen, Y., Xue, J., Zhang, B., He, L., & Chen, Y. (2024). Large-volume leo satellite imaging data networked transmission scheduling problem: Model and algorithm. *Expert Systems with Applications*, *249*, 123649.

Lo, M. W. (1999). Satellite-constellation design. *Computing in Science & Engineering*, *1*(1), 58–67.

Macarthur, K., Stranders, R., Ramchurn, S., & Jennings, N. (2011). A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 25, pp. 701–706.

Maheswaran, R. T., Pearce, J. P., & Tambe, M. (2004). Distributed algorithms for DCOP: A graphical-game-based approach. In *Proceedings of the International Conference on Parallel and Distributed Computing Systems*, pp. 432–439.

Mahmud, S., Choudhury, M., Khan, M. M., Tran-Thanh, L., & Jennings, N. R. (2019). AED: An anytime evolutionary DCOP algorithm. *arXiv:1909.06254*.

Mailler, R., & Lesser, V. (2004). Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 438–445.

Modi, P. J., Shen, W.-M., Tambe, M., & Yokoo, M. (2005). ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, *161*(1-2), 149–180.

Nag, S., Li, A. S., & Merrick, J. H. (2018). Scheduling algorithms for rapid imaging using agile cubesat constellations. *Advances in Space Research*, *61*(3), 891–913.

Netzer, A., Grubshtein, A., & Meisels, A. (2012). Concurrent forward bounding for distributed constraint optimization problems. *Artificial Intelligence*, *193*, 186–216.

NewSpace (2023). Newspace constellations. https://www.newspace.im. Accessed: 2024-08-12.

Nguyen, D. T., Yeoh, W., Lau, H. C., & Zivan, R. (2019). Distributed Gibbs: A linear-space sampling-based DCOP algorithm. *Journal of Artificial Intelligence Research*, *64*, 705–748.

Nguyen, T. T., & Yao, X. (2012). Continuous dynamic constrained optimization—the challenges. *IEEE Transactions on Evolutionary Computation*, *16*(6), 769–786.

Parjan, S., & Chien, S. A. (2023). Decentralized observation allocation for a large-scale constellation. *Journal of Aerospace Information Systems*, 1–15.

Pearce, J. P., & Tambe, M. (2007). Quality guarantees on k-optimal solutions for distributed constraint optimization problems.. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1446–1451.

Pertzovsky, A., Zivan, R., & Agmon, N. (2024). Collision avoiding max-sum for mobile sensor teams. *Journal of Artificial Intelligence Research*, *79*, 1281–1311.

Petcu, A., & Faltings, B. (2005). DPOP: A scalable method for multiagent constraint optimization. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 266–271.

Phillips, S., & Parra, F. (2021). A case study on auction-based task allocation algorithms in multi-satellite systems. In *Proceedings of AIAA Scitech*.

Picard, G. (2021). Auction-based and distributed optimization approaches for scheduling observations in satellite constellations with exclusive orbit portions. *arXiv:2106.03548*.

Planet (2023). Our constellations. https://www.planet.com/our-constellations. Accessed: 2024-08-12.

Pujol-Gonzalez, M., Cerquides, J., Meseguer, P., Rodríguez-Aguilar, J. A., & Tambe, M. (2013). Engineering the decentralized coordination of UAVs with limited communication range. *Advances in Artificial Intelligence*, *1*, 199–208.

Rachmut, B., Zivan, R., & Yeoh, W. (2022). Communication-aware local search for distributed constraint optimization. *Journal of Artificial Intelligence Research*, *75*, 637–675.

Salzman, O., & Stern, R. (2020). Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 1711–1715.

Schaffer, S., Chien, S., Branch, A., & Hernandez, S. (2018). Automatic orbit selection for a radio interferometric spacecraft constellation. *Journal of Aerospace Information Systems*, *15*(11), 627–639.

Shah, V., Vittaldev, V., Stepan, L., & Foster, C. (2019). Scheduling the world's largest Earth-observing fleet of medium-resolution imaging satellites. In *Proceedings of the International Workshop on Planning and Scheduling for Space*, pp. 156–161.

SpaceX (2023). How starlink works. https://www.starlink.com/technology. Accessed: 2024-08-12.

Squillaci, S., Pralet, C., & Roussel, S. (2023). Scheduling complex observation requests for a constellation of satellites: Large neighborhood search approaches. In *Proceedings of the International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 443–459.

Squillaci, S., Roussel, S., & Pralet, C. (2021). Managing complex requests for a constellation of Earth-observing satellites. In *Proceedings of the International Workshop on Planning and Scheduling for Space*.

Stranders, R., Farinelli, A., Rogers, A., & Jennings, N. (2009). Decentralised coordination of mobile sensors using the max-sum algorithm. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 299—304.

Swope, J., Mirza, F., Dunkel, E., Candela, A., Chien, S., Holloway, A., Russell, D., Sauvageau, J., Sheldon, D., & Fernandez, M. (2023). Benchmarking space mission applications on the snapdragon processor onboard the iss. *Journal of Aerospace Information Systems*, *20*(12), 807–816.

Tassa, T., Grinshpoun, T., & Zivan, R. (2017). Privacy preserving implementation of the max-sum algorithm and its variants. *Journal of Artificial Intelligence Research*, *59*, 311–349.

Troesch, M., Mirza, F., Hughes, K., Rothstein-Dowden, A., Bocchino, R., Donner, A., Feather, M., Smith, B., Fesq, L., Barker, B., & Campuzano, B. (2020). MEXEC: An onboard integrated planning and execution approach for spacecraft commanding. In *Proceedings of the International Conference on Automated Planning and Scheduling, Workshop on Integrated Execution / Goal Reasoning*.

Wang, X., Wu, G., Xing, L., & Pedrycz, W. (2020). Agile Earth observation satellite scheduling over 20 years: Formulations, methods, and future directions. *IEEE Systems Journal*, *15*(3), 3881–3892.

Yeoh, W., Varakantham, P., Sun, X., & Koenig, S. (2015). Incremental DCOP search algorithms for solving dynamic DCOP problems. In *Proceedings of the IEEE/WIC/ACM*

International Conference on Web Intelligence and Intelligent Agent Technology, Vol. 2, pp. 257–264.

Zhang, W., Wang, G., Xing, Z., & Wittenburg, L. (2005). Distributed stochastic search and distributed breakout: Properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, *161*(1-2), 55–87.

Zilberstein, I., Candela, A., Chien, S., Rijlaarsdam, D., Hendrix, T., Buckley, L., & Dunne, A. (2024a). Demonstrating onboard inference for Earth science applications with spectral analysis algorithms and deep learning. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space*.

Zilberstein, I., Rao, A., Salis, M., & Chien, S. (2024b). Decentralized, decomposition-based observation scheduling for a large-scale satellite constellation. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 34, pp. 716–724.

Zivan, R., Parash, T., Cohen, L., Peled, H., & Okamoto, S. (2017). Balancing exploration and exploitation in incomplete min/max-sum inference for distributed constraint optimization. *Autonomous Agents and Multi-Agent Systems*, *31*, 1165–1207.