Decentralized On-board Planning and Scheduling for Crosslink-enabled Earth-observing Constellations

by

Warren Grunwald

B.S. Mechanical Engineering, University of California at Berkeley (2012)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2019

Signature redacted

Author .....
Department of Aeronautics and Astronautics
August 22, 2019

Signature redacted

Approved by .....
Mark Abramson
Principal Member of the Technical Staff
The Charles Stark Draper Laboratory, Inc
Technical Supervisor

Certified by ..... Signature redacted .....
Kerri Cahoy
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Signature redacted

Accepted by .....
Sertac Karaman
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Decentralized On-board Planning and Scheduling for Crosslink-enabled Earth-observing Constellations

by

Warren Grunwald

Submitted to the Department of Aeronautics and Astronautics
on August 22, 2019, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Small satellites have improved in capability, nearing a future where high data-rate payloads and crosslinks can provide improved geospatial and temporal coverage, while at a fraction of the cost. Planning and scheduling for efficient bulk data routing with discrete crosslink windows in a dynamic network is a difficult combinatorial optimization problem [30]. As problem size grows, quickly solving the planning and scheduling problem involves implementing algorithms that can leverage parallelization. Decentralized algorithms are inherently parallelizable and can be implemented on-orbit by individual satellites.

This thesis investigates a decentralized approach that builds upon the Coupled Constraints Consensus Based Bundle Algorithm (CCBBA) with enhancements to address maximum flow problems. Maximum flow problems occur when moving some resource from sources to sinks across a network, such as a satellite constellation observing targets (sources), moving data between satellites with crosslinks, and downlinking to ground stations (sinks). The CCBBA enhancements include task forking, task outflow coupling, and dynamic task creation based on satellite flow direction preferences. These enhancements increase the total data throughput and decrease required runtime. When implemented on each satellite, this decentralized auction-based approach, named Iterative-CCBBA for Maximum Flow problems (ICMF), provides the following benefits: 1) has robustness in convergence to differences in agent situational awareness, 2) decouples operations from ground station planning resources, and 3) provides an inherently parallelizable algorithm, if implemented on the ground instead of each satellite.

ICMF is compared to a state of the art Centralized Global Planner (CGP) in six test cases, with two different inclinations and three different numbers of total satellites. Across all six unique use cases, ICMF has linear scaling in number of consensus rounds and, on average, runs in 94% less time than the CGP, with a 4% improvement in total data volume delivered. ICMF is an effective planner for satellite constellations that value total data throughput and runtime efficiency. The CGP performs better on median latency for observations and median average target

age of information, performing better by 58% and 23%, respectively. Future work options for incorporating additional data routing information that could help close the latency and target age of information gap while still using a decentralized approach are presented.

Thesis Supervisor: Kerri Cahoy
Title: Associate Professor of Aeronautics and Astronautics

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ADCS Attitude Determination and Control Subsystem

ADEPT All-Domain Execution and Planning Technology

AoI Age of Information

ASPEN Automated Scheduling and Planning ENvironment

AWS Amazon Web Services

BDRP Bulk Data Routing Problem

CBBA Consensus Based Bundle Algorithm

CCBBA Coupled Constraints Consensus Based Bundle Algorithm

CGP Centralized Global Planner

CLICK CubeSat Laser Infrared CrosslinK

CMF CCBBA for Max Flow

DARPA Defense Advanced Research Projects Agency

DV Data Volume

GA Genetic Algorithm

GPS Global Positioning System

ICMF Iterative CCBBA for Max Flow problems

LCCC Limited Communication Constellation Coordinator

LEO Low Earth Orbit

MILP Mixed Integer Linear Program

MIT Massachusetts Institute of Technology

NASA National Aeronautics and Space Administration

NEN Near Earth Network

NODE Nanosatellite Optical Downlink Experiment

OAP Orbit Average Power

PAT Pointing, Acquisition, and Tracking

RAM Random Access Memory

RX Reception Crosslink

SLO San Luis Obispo

SPRINT Scheduling Planning Routing Inter-satellite Network Tool

SSTP Small Spacecraft Technology Program

STAR Lab Space Telecommunications, Astronomy and Radiation Laboratory

TX Transmission Crosslink

vCPU Virtual Central Processing Unit

Symbols

To facilitate understanding of this thesis, some of the pseudocode will use an object-oriented style. The objects are included at the end of this symbol list, see Appendix B for definitions of all properties and methods.

N_u	total number of agents
N_t	total number of tasks
L_t	task bundle size limit
N_{steps}	total timesteps in the planning horizon
$\mathcal{I}$	set of all agents
$\mathcal{J}$	set of all tasks
$\mathcal{T}$	set of all timepoints in the planning horizon
$\mathcal{I}_j$	set of all agents that can bid on $Task$ with index j
$\mathcal{J}_i$	set of all $Tasks$ biddable by $Agent$ with index i
$\mathcal{J}_i^{exec}$	set of all $ExecutableTasks$ biddable by $Agent$ with index i
$\mathbf{x_i}$	assignment vector for $Agent$ with index i, $\mathbf{x_i} \in \{0,1\}^{N_t}$
$\mathbf{b_i}$	task bundle for $Agent$ with index i, ordered by decreasing task value
$\mathbf{p_i}$	task path for $Agent$ with index i, ordered by increasing task start time

$\oplus_{idx}$	insertion symbol. Inserts the argument on the right into the argument on the left at the location denoted by idx
$diam$	diameter of a communication networks; measures the maximum number of hops between any two agents in the network
$d_i(t)$	data state for *Agent* with index i at time t
$e_i(t)$	energy state for *Agent* with index i at time t
$a_i(t)$	activity state for *Agent* with index i at time t
D_{MAX}	maximum data storage on board a satellite
D_{MIN}	minimum data storage on board a satellite
$\dot{D}_{type}(a_i(t))$	data rate from activity at time t
$\dot{E}_{type}(a_i(t))$	energy rate from activity at time t
$\dot{D}_{J_e.type}$	data rate from activity of type $J_e.type$
$\dot{E}_{J_e.type}$	energy rate from activity of type $J_e.type$
$E_i(t)$	sunlight tracker for *Agent* with index i at time t: 1 for sun, 0 for eclipse
$\dot{K}_{chg}$	charging rate when in sunlight
I	*Agent* object, has agent-unique index i
J	$Task$ object, has task-unique index j
J_e	$ExecutableTask$ object, has executable task-unique fork number j_e
$E_v(e_i(t), J_e)$	data state violation from adding $ExecutableTask$ J_e to the bundle
$D_v(d_i(t), J_e)$	energy state violation from adding $ExecutableTask$ J_e to the bundle
R_c	number of communication rounds within the consensus phase
$\omega_{\mathbf{ij}}^{\mathbf{solo}}$	mutual dependence counting parameter for *Agent* with index i to bid solo on $Task$ with index j

ν_{ij}	timeout parameter for *Agent* with index i for $Task$ with index j
$iters_{MAX}$	ICMF algorithm parameter: (integer) sets number of CMF rounds that are iteratively executed
$TX_{multibid}$	ICMF algorithm parameter: (boolean) determines if new crosslinks created are multibid (True) or singlebid (False)
$J_e.forkable \in \{T, F\}$	object property example
$\text{fork}_{J_e}(newTaskInds)$	object method example

Chapter 1

Introduction and Motivation

The purpose of this thesis is to present an algorithm based on set of enhancements to the Coupled Constraints Consensus Based Bundle Algorithm (CCBBA) decentralized task allocation algorithm that can solve maximum flow problems that arise in Earth observing satellite constellations. Maximum flow problems involve moving a resource, such as observation data, from a set of sources to a set of a sinks across a network. The new algorithm, known as ICMF, is presented with simulation results for solving a Bulk Data Routing Problem (BDRP) with small satellites. Bulk data routing means transmission of raw data that is observed by any satellite in the constellation to any ground station in the network. The bulk data routing problem is solved with a feasible plan for each satellite in the constellation which optimizes for total data throughput, with a preference for lower average observation latency and lower average age of target information. The problem is described in more detail in Section 2.4.

1.1 Small Satellite Capability Growth

A 'small satellit' is any satellite that is less than 180 kg, according the National Aeronautics and Space Administration (NASA) Small Spacecraft Technology Program (SSTP) [37]. CubeSats will soon be capable of supporting high-data rate payloads, such as hyperspectral imagers [34, 59] and video [16], with laser communication crosslinks and downlinks [6, 44, 22] to achieve tens of Gb downlinked per orbit of valuable scientific data. Overall, the data delivered by small satellites is expected to

double from 2020 to 2030, reaching up to 3.9 exabytes [58] (exabyte = 10^{12} MB) over that decade. In addition to the cost and performance benefits, networks of satellites also have higher resilience against anti-satellite actions [12]. However, for small satellites to achieve this level of data throughput, the satellites need to coordinate their actions across the constellation.

1.2 Coordination of Small Satellite Constellations

Efficient coordination of many satellite actions in a constellation, such as SpaceX's proposed Starlink constellation [52], especially with mutually dependent tasks such as laser communication crosslinks, requires solving challenging planning and scheduling problems. Current methods involve framing the planning and scheduling problem as an optimization. Because these optimization problems are usually computationally taxing, schedules have been computed at ground stations and then uplinked to the satellites. Some missions only require observations and downlinks to be scheduled, such as the Planet Labs mission which uses CubeSats to image the entire landmass of Earth every day. Their approach uses simulated annealing to solve the optimization in a non-deterministic way [51]. Other non-deterministic methods have been demonstrated in simulation. Zheng *et al.* showed a coordination approach for a constellation with intersatellite communication between one 'mother satellite' and multiple 'daughter satellites' using a genetic algorithm [62].

However, there are limited results for large constellations with crosslinks enabled between all satellites. Zhou *et al.* demonstrated a contact plan approach that could effectively plan for a six-satellite constellation in two near polar orbits that scaled well with planning horizon, but did not show scaling results as the constellation size increased [65]. Scaling effectively with the number of satellites is critical for crosslink-enabled constellations because the number of possible routes that data can take from the observation point to a downlink opportunity scales at least exponentially with the number of satellites because the number of simple paths in the communication graph grows factorially in the worst case [24]. Andrew Kennedy's 2018 PhD thesis addressed this problem by using a route downselection phase to limit the number of

routes considered per observation to a constant value [30]. Kennedy demonstrated that the problem could scale polynomially, if parallel processing is used for the route downselection [29]. However, both Kennedy's and Zhou's methods require all of the problem information to be in one location and to have sufficient computational resources in that location. The centralization of information and processing power provides a large roadblock to implementing these methods on-orbit.

1.3 Motivation for Decentralized Planning and Scheduling

Using a decentralized approach for planning and scheduling has two main benefits. The first is that decentralization naturally works with parallelization because it is intended for use by multiple agents. If a constellation operator wants to implement the methods at the ground station, they could run a decentralized algorithm on parallel processors and reap the benefits of on-demand computation services, such as Amazon Web Services (AWS), more easily. However, the ultimate goal with decentralized methods is to deploy them in flight software on-orbit. Deploying the algorithms on-orbit could be done with centralized algorithms, but places a larger computational burden on one satellite instead of spreading it around the entire constellation. In most decentralized algorithms, each agent is executes a relatively simple set of procedures and the problem is solved by the combined effort of all agents.

This leads to the second main benefit of decentralized methods: the ability to execute the mission with minimal required ground station direction. This leads to many more specific positive aspects, such as:

- Lower ground station staffing requirements and no requirement to centralize all planning information in one location for each planning session.
- Robustness to correlated ground station failures.
- Ability to autonomously handle distributed user requests. If the constellation is serving a distributed user base that has tactical uplink terminals, then direct requests to the constellation may be more feasible then communicating back to

a ground station to incorporate a task into the next plan. The satellites can accumulate tasks on-orboard and use them in the next planning cycle. Or if the tasks are high priority enough, then they can trigger a new replan to serve the users more immediately and directly.

In this thesis, ICMF was developed from a general decentralized task allocation algorithm that can handle coupled constraints, known as CCBBA. ICMF introduces several key new enhancements to CCBBA [61] to increase performance and convergence time for BDRPs. The enhancements are: task forking, task outflow coupling, exact time matching, flow-preference sharing, and iterative new task creation. With these enhancements, ICMF is compared to a state of the art centralized planner in six test cases, resulting in 94% less runtime with a 4% improvement in total data volume delivered, averaged over the six cases.

The next chapter in this thesis starts with reference material on multi-agent planning and scheduling. Then max-flow problem and different solutions are summarized, since the BDRP is a specific instance of a max-flow problem. Chapter 3 describes the BDRP mathematically, presents the specific use cases that are tested, and describes ICMF in detail. Chapter 4 presents the results when varying the ICMF parameters and the performance against a state of the art centralized planner. Finally, Chapter 5 summarizes findings and describes areas for future work.

Chapter 2

Background and Literature Review

This chapter focuses on the capabilities of small satellites and the algorithmic background. First, small satellite capabilities and approaches for satellite constellation coordination are reviewed. Next, multi-agent planning and scheduling is reviewed, with an emphasis on how the problem can be solved with different levels of autonomy for the satellite. Then, decentralized approaches for planning and scheduling are discussed. Finally, ideal max flow problems and algorithms are presented in the context of laser communication enabled bulk data routing.

2.1 Small Satellites and Bulk Data Routing

This section will describe the capabilities of small satellites, focusing on the ability to route large of amounts of data with increased throughput, latency, and resilience.

2.1.1 Proliferation of Small Satellite Constellations

According to the NASA SSTP, a satellite is a 'small satellite' when its wet mass is below 180 kg. There are four main subcategories of small satellites in this range shown in Table 2.1 [37].

Name	Mass Range (kg)
Mini-satellite	500-100
Micro-satellite	100-10
Nano-satellite (CubeSat)	10-1
Pico-satellite	<1

Table 2.1: Small Satellite Categories by Mass [37]

Over the past ten years, small satellites have come to dominate the number of satellites launched, especially CubeSats. This thesis focuses on 6U CubeSats, which are still considered nano-satellites even though their mass can be to 12 kg [33]. CubeSats were proposed in 1999 by Bob Twiggs at Stanford and Jordi Puig-Suari at Cal Poly San Luis Obispo (SLO) with the development of a standard launcher and bus as a way to help university students build and launch satellites on a low-cost platform [47]. The first CubeSat launch occurred in 2003, and by the end of 2012, over 100 had been launched, mostly by universities [57]. As the technologies for CubeSats matured and the students working on them entered the work force, more commercial entities became interested in the capabilities of CubeSats. By 2014, the majority of CubeSat launches were commercial and there were over 40 new companies established aiming to leverage small satellite capabilities [58]. This planned growth of small satellite constellations also lead to increasing ground station and operator requirements to coordinate and schedule the growing number of satellites in orbit.

This thesis assumes that the satellites use the NASA NEN for ground stations. The NEN consists of ten ground stations around the world, some with multiple antennas, that can provide downlink and uplink support for satellite missions. NASA's analysis shows that the NEN has difficulty supporting equatorial CubeSats, but can support additional capacity for inclined and polar CubeSat constellations [53]. The ten NASA NEN ground station locations are summarized in Table 2.2. Note that some of these locations have multiple antennas with different abbreviations and that these are the same locations and abbreviations that will be seen in Chapters 3 and 4 of this thesis when discussing the test cases and results.

Station Location	Abbrev.	Lat (deg)	Lon (deg)
Fairbanks, Alaska, USA	AS1	64.8587	-147.8576
McMurdo, Antartica	MG1	-77.8391	166.6671
Wallops, Virgina, USA	WG1	37.9249	-75.4765
SSC South Point, Hawaii, USA	USHI01	19.014	-155.6633
SSC Kiruna, Sweden	KU1S	67.8896	21.0657
SANSA, South Africa	HB5S	-25.8869	27.7067
SSC Dongara, Australia	AUWA01	-29.0457	115.3487
KSAT TrollSat, Antartica	TR2	-72.0022	2.0575
KSAT Svalbard, Norway	SG1	78.231	15.389
KSAT Singapore	SI1	1.3962	103.8343

Table 2.2: NEN Ground Station Parameters [41]

To provide additional ground station services to the small satellite market, AWS has entered the ground station business in a joint venture with Lockheed Martin to provide ground station capabilities as a service [17]. The capability to have the data reception, processing, and storage handled by a separate entity could drive satellite and mission providers to develop on-board planning and scheduling capabilities so their planning segment isn't tied to a specific ground segment and they can pursue different options for ground station services without worrying about the planning and scheduling software compatibility. This is just one potential motivating factor for pursuing a decentralized planning and scheduling approach. However, to perform on-board planning and scheduling, the satellite bus has to be capable enough to execute the mission and carry the overhead for additional computation and communication.

2.1.2 Bus Capabilities

As CubeSats became more popular for university projects, commercial companies improved the capabilities of nearly every subsystem. For example, initially single junction solar panels with an efficiency of 16.9% were directly mounted to the surface, while now there are deployable solar panels with triple-junction GaAS solar cells

with efficiencies up to 33% [37]. Prices for the various subsystems has also decreased significantly over time due to the large market for CubeSat subsystems and the bus standard published by Cal Poly SLO [13]. Some vendors offer the entire CubeSat bus, with a specified payload interface for power, data, and mechanical so that the customer can focus on payload and mission applications. One example is the nano-avionics M6P 6U Satellite Bus [40].

The propulsion unit [39] takes up 1U of space in the M6P 6U Satellite Bus, so there is up to nearly 5U available for the mission payload and communications equipment when the propulsion unit is removed. By modifying that bus with additional solar panels similar to the deployable solar panels used by the MarCO mission, the orbit average power generation can reach up to 36 W [35]. By removing the propulsion unit, adding an X-band downlink antenna compatible with the NASA NEN [43], additional computer for planning and scheduling, and two additional battery cells, this thesis assumes the bus capabilities outlined at the end of this section in Table 2.3.

The capabilities in Table 2.3 are used to represent the state constraints in the planning and scheduling problem. The physical basis for the rest of the activity parameters used in this thesis, observations and crosslinks, are presented in Sections 2.1.3 and 2.1.4 and are reviewed in Section 3.1.2.

2.1.3 High Data Rate Payloads

Traditionally, high data rate payloads, such as hyperspectral imagers, required a large satellite bus and expensive launch vehicle. For example, the NASA Earth Observing 1 (EO-1) mission launched in 2000 and cost over $200 M to develop [34]. However, NASA has proposed a follow-on mission that leverages the recently developed Headwall Nano-hyperspec instrument, which can fit on a CubeSat and generates 0.5-1 Gbps of raw data, for a total mission cost below $5 million. While the best resolution is limited by the size of the CubeSat aperture, using a constellation of these CubeSats would provide much higher revisit time and more data at a fraction of the development and launch costs [34]. The NASA CubeSat hyperspectral imager is still in study phases, but with more integration effort, even better performance can be

achieved, such as the multi-spectral imager generating about 5 Megabits per picture (1 Mb per band) proposed by Tsitas and Kingston that fit on an 8 kg CubeSat and has similar payload and downlink capabilities when compared with a 150 kg satellite [59].

Gathering video from space is another mission that small satellites can soon fulfill. One company, Skybox Imaging, used 83 kg satellites to provide the first ever high definition video (above 50 Mbps data generation) from space [16]. This company was acquired by Planet Labs in 2017 [58]. Planet Labs operates a fleet of over 100 small satellites, mostly 3U CubeSats, to provide imagery of over 150 million km^2 every day [51]. As these examples demonstrate, there is a large and growing market for dense data products with improved latency. With the proliferation of CubeSats and the ability to fly high data rate payloads, the amount of data to be downlinked by small satellites is expected to double in the next decade, reaching up to 3.9 exabytes [58] over 10 years. This leads to a challenge for energy-limited and storage-limited CubeSats, because the data downlinked per unit energy is not yet high enough using UHF and S-band radios, as shown in Clements *et al.* [11].

For higher data rate payloads, CubeSats need to leverage more efficient downlink methods, such as X-band radio or laser communications. If a laser communication payload, which is also capable of cross-linking with other satellites, is used, this can increase total mission performance, because using crosslinks can achieve +25% throughput and up to 75% reduction in latency, which is discussed in more detail in Section 2.3.2 and Figure 2-1.

2.1.4 Laser Communications Crosslinks

Recent efforts in the Massachusetts Institute of Technology (MIT) Space Telecommunications, Astronomy and Radiation Laboratory (STAR Lab) have focused on developing and demonstrating laser communication crosslinks for CubeSats. First, Nanosatellite Optical Downlink Experiment (NODE) was developed to show the feasibility of an optical downlink on a CubeSat at a data rate up to 43 Mbps and demonstrated the feasibility of a low-cost laser communications ground terminal known as

PorTeL [11, 50]. Then, the CubeSat Laser Infrared CrosslinK (CLICK) CubeSat was developed to show the feasibility of closing a laser communications link between CubeSats, especially the Pointing, Acquisition, and Tracking (PAT) capability, and is capable of achieving up to 25 Mbps at 920 km range with a size of 1.5 U [6, 44, 22]. With the small form-factor of the CLICK payload, it is feasible to have a primary payload, such as a hyperspectral imager, on a 6U CubeSat as well.

Crosslinks make the optional planning and scheduling problem harder (NP-Hard) because optimal solutions require information about all possible paths between observations and downlink, referred to as data routes. The number data routes scales exponentially in the number of satellites since there are an exponential number of simple paths in an undirected graph [24]. Solving the crosslink-enabled bulk data routing problem efficiently is the main focus of this thesis and will be discussed in detail in Chapter 3.

By allocating 1.5U of payload space to a laser communications payload, 1U of payload space to the hyperspectral imager and processing electronics, and 1.5U of space to an additional planning computer, Global Positioning System (GPS) receiver, and Xband radio electronics, this thesis assumes the capabilities shown in Table 2.3 for a notional crosslink-enabled hyperspectral imaging 6U CubeSat.

2.2 Coordinated Small Satellite Constellations

Coordination in this context is defined as leveraging information about what activities other satellites plan on executing to determine what activities a particular satellite should execute. Coordination could occur on the ground, where a centralized planner has access to information about all the satellites, or on-board, where the satellites communicate with each other to adjust their plans. One example of a satellite constellation with minimal coordination is the GPS constellation, where each satellite is set to broadcast a signal repeatedly and the actions of the other satellites have little to no impact on the plan for that satellite. A planned constellation with high levels of coordination is the SpaceX Starlink constellation, where over 10,000 satellites must coordinate constantly for low latency routing decisions [23].

Subsystem	Capability
Payloads	Hyperspectral Imager and Laser Communications Crosslink
Power Generation (Orbit Average Power (OAP))	30 W
Power Storage	10 cells, 8.0 V, 13.8 Ah
Attitude Determination and Control Subsystem (ADCS)	3-axis stabilized, rate: 10°/s, control: ±0.1°
Orbit Determination	GPS Receiver (0.5U and 1W power) [55]
Bus base power draw	10 W
Data Storage	12 Gb Solid State Memory
Downlink Rate	-20 Mbps at 20 W total power draw
Observation Rate	+50 Mbps at 10 W total power draw
Crosslink Transmission	-10 Mbps at 20 W total power draw
Crosslink Reception	+10 Mbps at 5 W total power draw

Table 2.3: Assumed notional 6U CubeSat Subsystem Capabilities

2.2.1 Benefits of Coordination

As mentioned in Section 2.1.4, crosslinks can significantly increase the complexity of coordinating the activities of a constellation. Crosslinks inherently require coordination because a transmission and reception crosslink are mutually dependent on each other. However, even in the no-crosslink case, coordination can still be required to deconflict downlinks from multiple satellites to the same ground station at the same time since this thesis assumes that a ground station can only point to and receive data from one satellite at a time. This occurs when more than one satellite is in view of the same ground station, which will happen if the constellation has enough satellites or the satellites are clustered together. Since CubeSats are usually launched as secondary payloads, they are likely to be clustered together and must use differential drag or propulsion to space themselves out within the orbit. In either case, the constellation will have radio frequency interference and/or dropped data packets if there is not coordination between satellites on scheduling the downlink activities.

There are many other potential benefits of mission level coordination, including multi-modal sensing [14], distributed aperture radar missions [54], and reactive constellations [1, 38]. Multi-modal sensing missions, such as the Defense Advanced

Research Projects Agency (DARPA) BLACKJACK program, need to coordinate to have heterogeneous payloads collect on the same target at specific times to gather temporally correlated data [14]. For distributed aperture radar missions, such as the Air Force's TechSat 21 program, coordination is required to set the radar imaging baseline to achieve high resolution images [54]. Reactive constellations are those that coordinate to exploit new information gained by some satellites (or ground operators) to replan selectively with satellites that have fast slewing capabilities [1, 38]. For example, if a satellite observed Target A and did some initial processing that resulted in a higher priority that initially assumed, it would broadcast this information which might result in other satellites slewing to view Target A instead of their original plan. Truly reactive constellations require advanced autonomy to process the observed data accurately and reason correctly about the decision to abandon the current plan based on the new information. While there are other benefits of constellation coordination, this thesis will focus on coordination for the purposes of solving the bulk data routing problem and enabling future reactive constellation capabilities.

2.2.2 Planning Communication Architectures

To achieve coordination among satellites in a constellation, there are several communication architectures available for small satellites. In order of decreasing planning information latency, the communication architectures covered in this thesis are:

1. Ground Station Contact only: this is the traditional method of managing a satellite constellation, where planning information is only transmitted from a ground station when a satellite passes overhead. Assuming there is network connectivity between all of the ground stations, this allowed for a centralized planning algorithm to gather telemetry and payload data from all the satellites as they pass overhead and uplink new schedules. The coordination between satellites occurs indirectly in the centralized planning algorithm and the satellites do not have any direct information transfer between each other. This communication method is the simplest for the satellite software since there is no onboard planning or inter-satellite communication required.

2. Ground Station Contact plus Radio Crosslink Plan Dissemination: this method is the same as method 1, except that now satellites share new planning information with their neighbors via radio crosslinks. This approach was demonstrated to improve mission performance in average revisit time using a weak consensus mechanism in the Limited Communication Constellation Coordinator (LCCC) algorithm [29]. However, this approach increases satellite bus complexity and requires additional radio communications to be planned in the activity schedule and power budget, while also still requiring ground stations to develop the new plan.

3. Ground Station Contact plus Intersatellite Backbone Plan Dissemination: this method is the same as method 1, except that there is an assumed communications backbone constellation that the small satellites can also all interface with to receive planning updates. This method has very low plan dissemination latency since all the satellites in the constellation can receive the information shortly after it is uplinked to the communications backbone without scheduling radio crosslinks with each other. This has been demonstrated with CubeSats and the Iridium communications constellation with the PicoPanther satellite [52]. There have also been simulations showing that this type of architecture, where the mission constellation is at a lower altitude than the communications constellation, provides more robustness to total satellite failures and better performance for similar cost values [12].

4. Radio Crosslink only: In this approach, there is no dynamic planning information from the ground stations and the satellites have to develop and disseminate the plan with crosslinks only. Satellites still receive static planning information from the ground as a starting point. This static information concerns aspects of the mission that are not expected to change between planning horizons, such as the location of the ground stations and the observation targets. The satellites still have to deconflict mission data downlinks to ground stations and let the ground station know which satellite has a downlink scheduled at each time

point. This has the highest complexity requirement on the satellite, but it frees up ground station planning resources for other constellations and also means that the satellite constellation is robust to ground station failures. This setting naturally lends itself to decentralized planning algorithms, but they must be able to handle the complex, coupled constraints of satellite missions with laser communications crosslinks. Since the satellites are developing the plan themselves, as soon as it is updated and consensus is achieved, all of the satellites already know the plan so there is no plan dissemination latency that would be added ontop of plan generation time since the time between plan convergence and dissemination is zero. Note: that does not mean that the replan time is zero since a disruption that requires a replan will require a new consensus to be achieved.

2.3 Multi-agent Planning and Scheduling

Multi-agent planning and scheduling covers a large set of potential applications. Traditionally, planning was done in a central location with all of the information about the problem represented in a mathematical model that could be solved. A planner that uses all of the problem information at once and then directs all of the agents is known as a centralized planner. For satellites, the location of the centralized planner is generally on the ground, but it could also be a special satellite on-orbit. If the satellites can decide on their own activities, but don't communicate directly with each other, instead going through one information hub satellite for coordination, then this is referred to as a semi-centralized approach because there is a still a single location where all information passes through. A truly decentralized approach has all satellites communicating directly with each other and the communication network is coupled with the agent's path (or each satellite's orbit, in the satellite network case).

2.3.1 Trade-offs in Different Approaches to Coordination

The fundamental trade-off between centralized, semi-centralized, and decentralized planning and scheduling algorithms is based on where the required information and

computing resources are located and how they interact during planning. For all three types, this thesis reviews limited planning horizon algorithms, where a specific time into the future is planned for. The difference between the three approaches can be summarized as:

- Centralized algorithms aggregate all information and use it one shot to solve the problem for a given time horizon. Because of this aggregation, there can generally be optimality guarantees for deterministic centralized methods, although the solutions are usually intractable because solving the BDRP with routing for individual observations, not just routing for bulk data, is NP-hard. So heuristic or randomized methods are usually employed to selectively throw away portions of the aggregated information. Both types of algorithms, heuristic and randomized, are discussed in Section 2.3.2.

- Semi-centralized algorithms still aggregate all information in one place, but the information might not all be there at the same time and the decisions could be made in a distributed way. This is the most general category, meant to catch the planning approaches that fall between centralized and decentralized. Optimality guarantees depend on whether information consistency is enforced, such as in the implicit coordination algorithm with perfect information [2]. Approaches that have demonstrated results for satellite networks are discussed in Section 2.3.2.

- Decentralized algorithms rely on agents communicating directly with each other and sharing some subset of information relevant to the planning problem. The agents that share information depend on the communication network at that time, which can change dynamically. Because these algorithms are often implemented on agents with limited processing power and energy storage, efficient computation and bounded messages between agents are both essential. There is a trade-off in guaranteeing convergence vs. optimality in the presence of information inconsistency due to noise which will be discussed more in Section 2.3.3

Centralized and some semi-centralized algorithms can directly enforce global information consistency. Since all information is available they can use statistical methods to perform a best estimate of the true global state if there are disagreements between local information received from agents. Because of the guarantee of globally consistent information, these algorithms can make optimality guarantees that are not available if global information consistency is not enforced [27]. Use and consistency of knowledge also directly impacts the routing capabilities of algorithms, especially in delay tolerant networks. Jain *et al.* investigated this trade off in knowledge about the network topology vs. routing performance and found that, in general, using the least knowledge led to the worst performance. However, there were situations where using limited additional knowledge could greatly increase performance [25]. This bodes well for decentralized algorithms that can create activities that require routes, such as ICMF, and determining the right information to share to increase this performance opens an interesting avenue of future work.

For all of these different approaches, especially if implemented on-orbit, another important aspect to consider is how planning duration can impact plan execution. If the planning duration is not accounted for and the planning duration runs into the start of the planning window, then the returned schedule can be infeasible and result in a cascade of schedule failures. One approach is just to the algorithm can always return a feasible solution, so that if the expected execution start window of the plan is reached then the agents can switch to execution. However, this can be difficult to implement and still maintain performance. Another approach, known as Deadline Aware Search, directly incorporates the deadline into the planning process and prunes solutions based on the distance to the deadline [15]. This can work well for fixed deadlines, but if the deadline is more flexible, then it leads to ambiguity on the benefits of continuing to plan vs. starting to execute. A more recent approach from Cashmore *et al.* in 2018 tries to estimate the trade-off in starting execution vs. continuing to plan, which allows for more flexible planning windows [8]. While the consideration of planning time on performance is important for an on-orbit planner that can solve the BDRP, this thesis does not directly incorporate it into the methdology. There

are areas outlined for future work in Section 5.2.2 that consider this issue.

2.3.2 Centralized and Semi-Centralized Approaches

This subsection covers the centralized and semi-centralized approaches that have been used to coordinate satellite activities. First, methods that use heuristics to prune the solution space are covered, with additional time spent on reviewing the SPRINT Centralized Global Planner (CGP), which is a heuristic centralized planner based on the GP-Fast algorithm developed by Andrew Kennedy [30]. This additional review is warranted because the SPRINT CGP is used for comparison with the performance of ICMF across six different test cases in Chapter 4. Then, methods that use a randomized approach are discussed.

Heuristic Methods

Heuristic methods for optimal planning and scheduling often involve some sort of local search that deliberately prunes the search space to reduce the runtime and/or memory required to solve the problem. Historically, satellite schedules have been planned by satellite operators working directly with collection requirements. Before discussing satellite constellation planning, a heuristic planning method used to partially automate (humans in the loop) single satellite operations is discussed. This method is part of the Automated Scheduling and Planning ENvironment (ASPEN) system used by NASA for the Citizen Explorer (CX-1) satellite and for automating ground station contact planning for the Deep Space Terminal (DS-T) [9]. The core of the planning and scheduling approach is known as iterative repair. The human operator can specify high level goals (or directly provide a low level schedule) and then the ASPEN system approach produces a base schedule and then iteratively improves the schedule by removing schedule conflicts (constraint violations) as time allows [48]. The iterative repair mechanism continues to run as long as planning time is available. While this mechanism is effective for single satellite operations, the local search heuristic does not provide guarantees on conflict resolution and it assumes operator interaction to perform effectively. This type of approach is a classic example of software enhanced satellite planning, but it does not scale well to managing constellations

of tens to hundreds of satellites, with coupled constraints between those satellites.

The classic planning problem for satellite constellations involves maximizing the value of observations and scheduling of downlinks, with no crosslinks between satellites. This problem area is still sufficiently rich because there can be a wide variety of additional constraints, such as consecutive target observations, temporal constraints, and different polygon shapes of target areas that require the planning algorithm to partition. One solution to this type of problem is to use a tabu search [19, 20], which is a type of local search that allows one step degradations of performance and uses a tabu list to prevent coming back to poor solutions, along with certain constraint relaxations to help explore the space faster. A modified tabu search was used by Bianchessi *et al.* in a small satellite constellation setting with multiple potential users, where observation value is analogous to profit. Their results achieved near optimal performance for two satellites with four different users over a 24 hour planning horizon with 10 minutes of planning time [5]. However, the limited constellation size and lack of description of computational complexity scaling with constellation size make it difficult to access performance in the BDRP context.

A recent advance that is directly applicable to the problem setting of this thesis, is the Global Planner from Andrew Kennedy's 2018 PhD thesis [30]. Kennedy provided a centralized planning method that could handle the increased computational complexity of crosslinks between satellites. The complexity increases from the total number of routes in the observation and downlink only case is $O(N_{dlnks}N_{obs})$, up to $O(N_{dlnks}N_{obs}N_u!)$ when crosslinks are included (Note: $N_u!$ here is number of satellites factorial).

Before discussing the global planner, some setup is warranted because the same setup is used for the approach in this thesis. The Global Planner, and the algorithms developed in this thesis, assume a starting point of knowing each satellite's physical access windows over the planning horizon. These windows come from knowing the target set, ground station locations, and other satellites' propagated orbital position over the planning horizon. Each satellite's orbit is propagated over the planning horizon and the following windows, which map to potential activities, are output:

- Observation Activity: Each satellite has a unique set of observations for the targets that are within their field of regard during the planning horizon. A single target can have multiple observation activities across multiple satellites, or the same satellite if the planning horizon consists of several orbits.

- Downlink Activity: Each satellite has a unique set of downlinks to the ground stations that are within their field of regard during the planning horizon. A single ground station can have multiple downlink activities across multiple satellites, or the same satellite if the planning horizon consists of several orbits.

- Crosslink Activity: Communication access between satellites when they can close a link. For satellites that are neighbors in the same orbital plane, this can be a continuous window which will be broken up into smaller fixed duration windows that can be scheduled selectively. A single satellite schedules either a transmission (TX) or reception (RX) crosslink, which is only valid if the corresponding satellite schedules the opposite crosslink type at the same time.

The Global Planner solves the BDRP in centralized manner with three steps:

1. Route Downselection: The information about the BDRP in the planning horizon is collected, which includes all satellite observation, downlink, and crosslink activities. These activity windows are used to create all possible data routes which consist of an observation, zero or more crosslinks, and a downlink. These data routes are downselected to keep a constant amount of routes per observation window using three heuristics (each configurable, with default values shown below):

    - h_{DV} (heuristic for data volume): Top 6 data throughput routes per observation. This ensures that that each observation activity has options for maximum data throughput.

    - h_{lat} (heuristic for latency) : Top 6 best (minimum) latency routes per observation. This ensures that each observation activity has options for best latency performance.

- h_{LO} (heuristic for least overlap): Top 30 least overlap routes per observation. This ensures route diversity to provide the activity scheduler with more options for an optimal schedule.

2. Model Construction: After route downselection, there will be at most $N_{obs} * (h_{DV} + h_{lat} + h_{LO})$ possible routes. These routes limit the number of possible activities that can be scheduled. Activities are constructed in a mathematical model as a Mixed Integer Linear Program (MILP). Constraints such as data storage, energy storage, and activity time limits are enforced via the MILP. In the implementation, this model construction occurs with the python optimization modeling software, Pyomo.

3. Activity Scheduling: Once the model is created in Pyomo, it is solved as a MILP using an optimization software package, such as Gurobi. The MILP is solved with objective function weights on data throughput, data latency, and energy margin. These can be tuned by each user, but are set by default to 1.0 to value them equally. The MILP solver itself uses heuristic methods to find a near optimal solution, but maintains an optimal tolerance bound by evaluating the dual formulation of the problem [30].

The basic strategy of the Global Planner is to reduce the problem size by cutting off activities that are likely to not be beneficial. The activities are cut off by examining the potential routes available in the problem with the heuristics shown above (h_{DV}, h_{lat}, h_{LO}). Using these heuristics for route downselection resulted in the Global Planner running two orders of magnitude faster for long planning horizons (above 1000 minutes) while maintaining 90% optimality [30]. Each of these steps can take a large amount of time for large problem sizes, but the key insight from the thesis was to make route downselection parallelizable to improve scalability with number of satellites [30]. This limits the complexity from adding crosslinks to a constant factor multiple of the number of satellites and targets, which makes planning for a constellation with crosslinks enabled more tractable. The benefits of utilizing crosslinks to solve the BDRP are summarized in Figure 2-1.

#	Metric Description	Goal	Change from Dlnk Only -> Dlnk + Xlnk
1	Obs. throughput (% of potential)	Max	**Increased from 44% to 70%** (Walker)
2	Median observation latency	≤ 10 min.	**Reduced from 63 to 15 minutes** (SSO Ring) **Reduced from 14 to 8 minutes** (Walker)

Figure 2-1: Performance benefits of adding crosslinks [30]

The Global Planner from Kennedy (2018) is used as the basis for a piece of open source software known as the Scheduling Planning Routing Inter-satellite Network Tool (SPRINT). The Global Planner from (Kennedy, 2018) was renamed the SPRINT CGP. The SPRINT CGP has a benchmark case to use the 10 NASA NEN ground stations, with 20 observation targets and 30 satellites in a 3 plane Walker configuration, inclined at 30°. However, there was an initial version of the problem that had 100 observation targets spaced equally around the world from -30° to 30° in latitude. This initial version served as the developmental test case for ICMF. All of the comparison cases for ICMF share the same orbit propagation and communication link modeler front end, so that the use cases are as similar as possible up to the start of the planning algorithm execution. These modules are shown in the context of the SPRINT pipeline in Figure 2-2.

Randomized Methods

In addition to heuristic methods, algorithms that leverage randomization techniques have also been used to solve satellite constellation scheduling problems. The company Planet Labs uses a scheduler known as micromanager which takes in available events (similar to observations and downlinks) and uses a simulated annealing algorithm to yield a good schedule [36]. This is done at the ground station and the schedules are uplinked to satellite as they pass overhead, with no intersatellite backbone to help plan dissemination. The simulated annealing algorithm was first proposed in 1983 by Kirkpatrick *et al.* Simulated annealing creates a connection to finding optimal energy states in statistical mechanics to finding the optimal solution in a combinatorial optimization problem, and leverages a randomized approach to approximate an optimal solution [31]. This approach is able to effectively solve the satellite con-

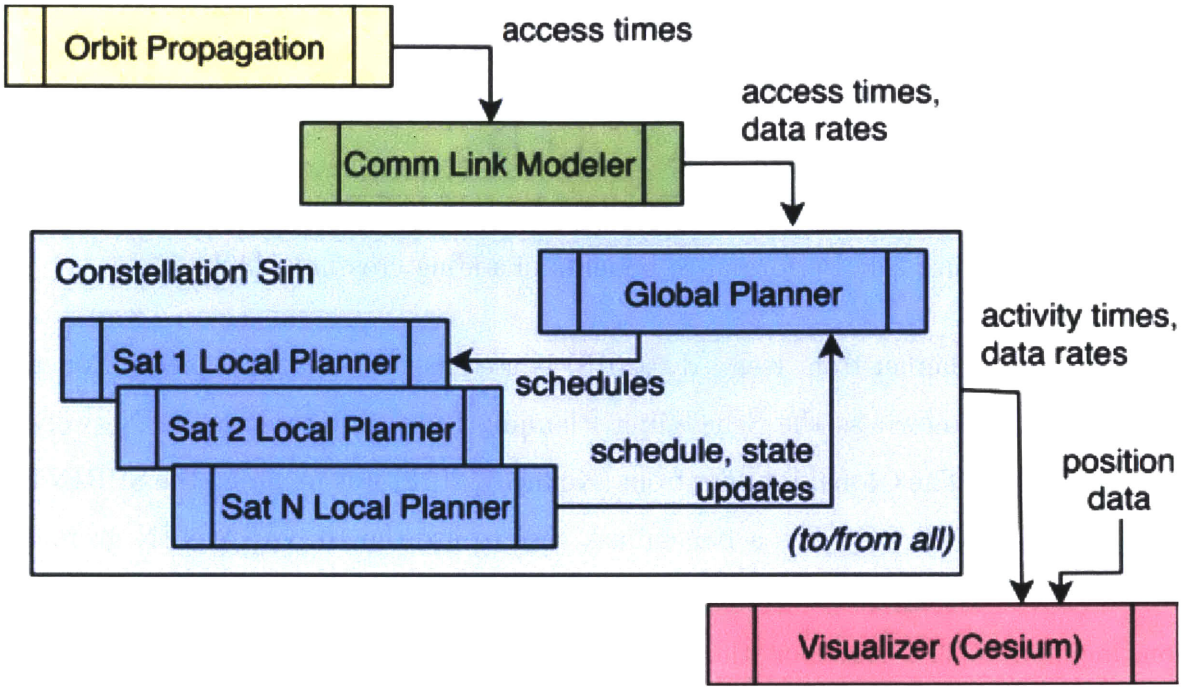


Figure 2-2: Computation and simulation pipeline used in SPRINT [30]

stellation scheduling problem for just observations and downlinks, but has not been applied to the BDRP with crosslinks enabled. This would likely be difficult because the simulated annealing approach makes randomized moves by shifting the schedule around. This activity would be more difficult to frame for crosslinks; to capture the mutual dependence between two different satellites, they would have to coordinate random moves to avoid breaking their respective schedules.

Another popular randomized approach is to utilize a Genetic Algorithm (GA). In general, a GA randomly tests solution points in the space and uses a fitness function to evaluate each point. Points must meet some fitness threshold to survive into the next round, and then they are combined in different ways to yield new points in the next generation. Different variants of GAs come with different mutation approaches when the points are combined. In one example of a semi-centralized approach that had multiple satellites coordinate their schedules via one central satellite, Zheng *et al.* developed a Hybrid Dynamic Mutation (HDM) strategy to coordinate actions between eight daughter satellites and one mother satellite [62]. Only one daughter satellite could communicate with the mother satellite at a time, and the mother satellite was

responsible for downlinking all data to Earth while the daughter satellites made observations. The HDM approach was shown to be the best approach on average for overcoming local optimum and runtime performance limitations among different GA variants for creating a schedule in this case. Zheng *et al.* later expanded their approach to include repair replanning mechanisms with a semi-centralized approach where the re-planner lives on the mother satellite, but is triggered by a status monitor on-board each daughter satellite [63]. Zheng *et al.* proposed two replanning techniques, both of which can successfully recover from emergency situations in simulation and trade-off data throughput vs. re-planning runtime. While Zheng *et al.* demonstrated effective recovery in a small scale constellation, runtime performance as a function of constellation size is not available and using a GA for completely decentralized planning, instead of just limited replanning, may struggle to provide guaranteed runtime bounds to come up with a feasible plan.

2.3.3 Decentralized Approaches

For any decentralized algorithm, one of the first things to determine is if a synchronous communication model can be used or an asynchronous model must be used. A synchronous model assumes that all operations occur in lockstep across the network, while an asyncrhonous model has no guarantee on the relative exeuction rates of processes and/or communication channels [24]. Relying on a synchronous model can make the analysis of the algorithms bounds on execution rounds and the number of messages more straightforward, and certain guarantees are possible, such as the ability to come to consensus in the presence of failures, that the asynchronous model does not allow [24]. This thesis primarily considers synchronous decentralized algorithms. Some asynchronous algorithms are considered as well for comparison. Since Low Earth Orbit (LEO) satellite clocks can be kept in sync with GPS, and as long as the clock difference and communication time between each agent is bounded, then the synchronous model is appropriate [32].

Before focusing on CBBA, which is an auction algorithm that forms the basis of ICMF, this thesis presents some other decentralized algorithms that have been

used in satellite planning and scheduling or data routing. Zheng *et al.* developed a fully distributed version of their Hybrid Dynamic Mutation GA approach that uses a local constraint satisfaction module to solve local feasibility before exchanging information to globally optimize the schedule directly among daughter satellites by contributing elements of their local population to the global population solution [64]. Their distributed randomized approach showed improvement in computation time once the problem duration increased to above 50 orbits (around the moon, in this case). The computation benefit increases as the number of satellites grows from 6 to 12. However, the computation time for both Zheng *et al.* centralized planner and the decentralized planner appears to grow exponentially as a function of the number of satellites. Because of this exponential runtime growth, randomized decentralized algorithms were not chosen for further investigation in this thesis.

Another interesting aspect of decentralized algorithms involves the field of mechanism design [42]. In general, the field of mechanism design focuses on how private preferences (not shared with other agents) can be steered toward choices that are beneficial for all agents in the problem. An important example of mechanism design is the border gateway protocol (BGP) used by different internet service providers to route data, with different quality of service requirements, across the internet seamlessly [42]. Mechanism design may become increasingly useful as satellite constellations managed by different users interact with each other in a federated network to provide services to end point users. Some work in multi-path, multi-hop routing has improved upon the mechanisms in BGP [56]. However, because the problem setting for this thesis involves full knowledge of all agent's preferences, mechanism design itself was not implemented.

Auction Algorithms

Algorithms that allow different agents to bid on certain tasks against each other are known as Auction Algorithms. This thesis focuses on combinatorial auctions, in which economic efficiency, and thus total global performance, is improved if bidders can bid directly on different combinations, or bundles, of tasks instead of just individual tasks

[60]. Combinatorial auctions have additional advantages in a dynamic communication network with limited communication between all agents because some conflicts can be resolved in parallel amongst the different agents. Single task auctions can be more effective with constant communication, such as the case of robots working directly next to each other. The MURDOCH system is one example of a single task auction approach that uses anonymous broadcasts between tightly coordinating robots to achieve global goals [18]. However, the satellite constellation problem setting does not have continuous communication between agents, so information takes time to propagate through the network and the number of agents can be over 100 for large constellations.

Decentralized combinatorial auctions methods were reviewed because bidding against bundles of tasks reduces the computational complexity of resolving the bidding and clearing portions for the auction. For certain subsets of bundles considered and the objective functions used, some bounded optimality guarantees can also be preserved [3]. This review of auction algorithms led to focus on decentralized task allocation algorithms that consider bundles of tasks and can be used in a dynamic communication network. Consensus Based Bundle Algorithm (CBBA) is one such algorithm that was developed by Choi *et al.* in 2009, and has had many variants and extensions since it was first developed for use in aerial drone task allocation [10].

CBBA

CBBA has two main phases that are repeated until no agent changes their bundle. The first phase is the bundle phase, in which all agents independently act to allocate tasks greedily to their bundle. The greedy allocation means that the next best value task (highest score function) is added to the bundle, without consideration for future combinations of tasks. CBBA assumes the score function is *Diminishing Margin Gain* (DMG), which means that no additional tasks can be added which would increase the score of existing tasks in the bundle. Johnson *et al.* elaborate on the importance of submodularity and how to handle non-submodular objective functions [26]. The second phase of CBBA is the consensus phase, where agents communicate with their

neighbors and fixed rules are used to reset tasks in the bundle to prevent conflict between agents in the global task allocation.

Since each satellite is assumed to have the capability to propagate their own orbit over the planning horizon and the starting states have uncertainty in them, each satellite's local information about when it has access to other satellites, ground stations, and targets will likely be inconsistent with the truth. Because of this local information inconsistency, there could be convergence delays if agreement on a shared truth state was required. However, some decentralized auction-based task allocation algorithms, such as CBBA [10], do not require information consistency between agents. Convergence time, measured in number of communication rounds, is not a function of relative information error. This is important, because it allows satellites to set a bound on communication required to come to a solution regardless of starting errors. There is a drop in optimality as starting error increases, but this is expected for most algorithms. The key take-away is that as long as the planning time and communication rounds can be effectively bounded, then an on-orbit decentralized planner is feasible.

This convergence guarantee is what led to the decision to base ICMF off a CBBA variant. Because of this, additional review of the mathematical formulation and generalized constraints is warranted. The objective function for CBBA is shown in Equation (2.1) with the generalized constraints immediately following.

$$\max \sum_{i=1}^{N_u} \left(\sum_{j=1}^{N_t} c_{ij}(\mathbf{x}_i, \mathbf{p}_i) x_{ij} \right) \tag{2.1}$$

subject to

$$\sum_{j=1}^{N_t} x_{ij} \leq L_t \qquad \forall i \in \mathcal{I}$$

$$\sum_{i=1}^{N_u} x_{ij} \leq 1 \qquad \forall j \in \mathcal{J}$$

$$\sum_{i=1}^{N_u}\sum_{j=1}^{N_t} x_{ij} = N_{min} := \min\{N_t, N_u L_t\}$$

$$x_{ij} \in 0,1 \qquad \forall (i,j) \in \mathcal{I} \times \mathcal{J}$$

where N_u is the number of agents, N_t is the number of tasks, L_t is the maximum bundle lengths (max number of tasks that an agent can assign), $\mathbf{x}_i$ is the vector of tasks that agent i plans to execute, where the elements of this vector are the decision variables, x_{ij}, relevant to agent i , $\mathbf{p}_i$ is the path vector which contains the task index and when agent i will execute that task, $c_{ij}(\mathbf{x}_i, \mathbf{p}_i)$ is the generalized score function.

In CBBA, convergence can be assured when an agent hasn't witnessed any changes to the consensus dictionaries for 2**diam* rounds, where *diam* is the communication network diameter, defined as the maximum number of hops between any two agents. So the algorithm keeps running until each agent observes this and then the algorithm terminates and/or moves on to another step. Without another consensus mechanism, agents running CBBA could declare termination at different times, but each time a node sees no changes for 2**diam* rounds ,then the entire algorithm has actually terminated and no agent will see more changes.

CBBA was originally developed for a synchronous communication environment, although all agents did not need to be in constant communication. Since it was originally developed, many other variants that are relevant to satellite network planning problem have been introduced, such as:

- Asynchronous CBBA (ACBBA): Developed by Johnson *et al.*, ACBBA contains the notions of both global and local convergence and allows agents to build bundles and perform consensus rounds on their own schedules [28]. This variant is appropriate if there is intermittent communication resulting in formation of a disconnected graph and/or limited ability to keep clocks synchronized.

- Ponda *et al.* developed a chance-constrained variant that uses a risk adjustment method based on individual agent risk distributions to ensure a mission level risk threshold is reached [46]. This method could be relevant to the satellite constellation planning scenario when certain targets and/or payloads have a probability of activity failure (*i.e.* due to cloud cover) and some of targets must be observed successfully at least once during a planning horizon with a specified probability of success.

CCBBA

The variant of CBBA that is most relevant to the BDRP is known as CCBBA because it assumes a synchronous communication mode and can handle complex mission constraints, such as temporal and mutual dependence constraints [61]. The ability to handle temporal constraints is required for the BDRP because satellites must execute their downlink activities only when all other satellites are not executing a downlink activity to the same ground station. Handling mutual dependence between tasks is what allows transmission and reception crosslinks to pair between different satellites.

Task coupling constraints are handled in CCBBA by forming a single activity between all tasks that share a coupled constraint, such as mutual dependence. All tasks in an activity, k, have their inter-dependencies represented in a dependency matrix, D^k. For an individual entry, D^k_{qu}, the following entries represent the relationship:

- 1 if task u depends on task q
- 0 if task u can be assigned independently of q
- -1 if task u and task q are mutually exclusive.

For the BDRP, the dependency matrix is only required for crosslinks, which are mutually dependent, so the dependency matrix will look like the following:

$$\mathcal{D}^{j\in\mathcal{J}_x}_{RX-TX} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.2}$$

where $j \in \mathcal{J}_x$ represents the fact that this dependency structure exists for all crosslink pairs. CCBBA allows mutually dependent tasks to be allocated by having optimistic vs. pessimistic bidding strategies. An optimistic bidding strategy means that the agent can bid on the task, even though all dependencies are not met. There are technically two separate levels of optimistic bidding, as shown in Figure 2-3, where the vector $\mathbf{w}_i^{solo}$ tracks if agent i can bid on tasks without any of their dependencies met and the vector $\mathbf{w}_i^{any}$ tracks if agent i can bid on tasks with at least one other dependency met. The elements of this vector are decremented every time a timeout parameter for each task j, ν_{ij}, is reset, which counts the number of rounds that the task has not met all of its dependency constraints. The authors note that usually elements of $\mathbf{w}_i^{any}$ are set higher than the elements of $\mathbf{w}_i^{solo}$. The multiplication of each ν_{ij} and corresponding element in $\mathbf{w}_i^{solo}$ yields the total number of rounds that an agent can bid solo on a given task. Increasing these values means that the agent can wait longer for another agent to bid on the coupled task, which increases convergence time, but can also increase performance.

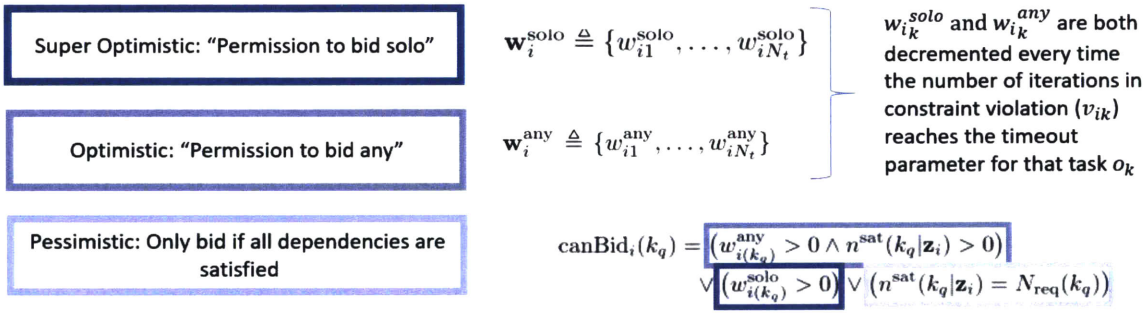


Figure 2-3: Summary of how CCBBA handles mutual dependence

Note: that the term 'Optimistic' is used to describe when both or either the 'solo' and 'any' bidding parameters are above zero in the CCBBA paper [61], and the term 'Super Optimistic' is only used in this thesis to help differentiate the two parameters.

CCBBA forms the basis for the algorithms described in this thesis. The enhancements to CCBBA to address the maximum flow elements of the BDRP are described in detail in Section 3.2.

2.4 Max-Flow Problems

This section will cover background information for network flow problems. Typically, the objective is to find the set of choices that permit maximum flow in the network from the sources to the sinks. These network flow problems are called maximum flow (max-flow) problems. Time-varying maximum flow problems include those where the properties of the network, such as edge capacity and connectivity, can change with time. Typically, centralized algorithms are used to solve these problems, and can do so in polynomial time [7]. The satellite BDRP is an instantiation of a resource constrained time-varying max-flow problem with the satellites as the network nodes, observation targets as the sources, ground stations as the sinks, and crosslinks are the inter-network connections between agents. The choices are which activities each satellite should execute to globally route the most data from the sources to the sinks. However, the BDRP does not translate directly to a typical time-varying maximum flow problem because of the addition of the energy state constraint, which limits the set of actions that can be taken in the planning horizon and is unique to each satellite because of different eclipse times.

2.4.1 Decentralized Max-Flow Approaches

For the synchronous decentralized algorithm setting with a fixed (not time-varying) network, auction algorithms can be used to solve the max-flow problem. Bertsekas shows that the problem can be reframed as a minimum cost auction problem and solved in $O(n^3)$ time [4]. This auction approach differs from the centralized approach of Goldberg and Targan, known as pre-flow/push, that could solve the problem in $O(nm \log(n^2/m)$ time and can also be parallelized [21] . The pre-flow/push algorithm allows an original estimate of flow, the pre-flow, to exceed capacity and then creates a residual graph that each node pushes excess flow towards the sink. This idea helped inspire the task-outflow coupling enhancement of ICMF described in Section 3.2.3. However, this algorithm in its original form assumed one single source and one sink in a centralized computation, so direct implementation was infeasible.

The preflow/push algorithm from Goldberg and Targan also inspired a fully de-

centralized and asychronous maximum flow algorithm developed by Pham *et al.* [45]. This algorithm can use multiple sources and sinks and each node performs local computations, then each node exchanges messages with each other until the maximum flow is established. The algorithm starts with a non-optimal flow across the network and terminates once no more flow can be pushed toward the sink. Time complexity is $O(n^2)$, while number of messages sent is $O(mn^2)$. This algorithm is promising for decentralized max-flow problems; however, it does not directly address time-varying problems and would need to also be augmented to handle total activity set constraints, such as a satellite energy state.

2.4.2 Implementing Max-Flow Algorithms for Small Satellites

While decentralized maximum flow algorithms exist and can address the core aspects of the BDRP, they do not directly address all the differences and model assumptions required for small satellite constellations, namely:

- Multi-source (observation targets), multi-sink (ground stations) problems with dynamic network connectivity;
- Ground station accesses need to be deconflicted in time;
- Crosslink mutual dependence: flows between network nodes need to be coordinated and agreed to (cannot send flow to another node without the other node also agreeing to accept flow at the same time);
- Energy state is another constraint, bin addition to flow capacity;
- Discrete problem with transition times in-between flow changes.

As a starting point for ICMF, a max-flow algorithm could have been used instead of CCBBA. However, this would have over-specified the algorithm since the goal is to develop an algorithm that performs well in the BDRP, but can handle other types of missions as well. This is why CCBBA was instead modified with some of the max-flow algorithm principals in mind.

Chapter 3

Approach

This chapter describes the decentralized approach used to solve the bulk data routing problem for small satellites.

First, the problem is framed in more detail, with a review of key assumptions, mathematical formulation, and explanation of the use cases that serve as the performance testing benchmarks. Then, the enhancements to CCBBA required to address maximum flow problems are presented in detail as the basis for a new algorithm called CMF. After the CCBBA enhancements are presented, Section 3.3 explains the main contribution of this thesis: the ICMF algorithm, which includes an iterative loop around CMF. Finally, a simple routing algorithm is presented, which uses the output of ICMF to generate data routes after the satellite's activities have been agreed upon.

3.1 Framing the problem

This section frames the bulk data routing problem in detail by enumerating key definitions and assumptions, providing the mathematical formulation of the problem and the relevant constraints, and explaining the use cases to test the algorithm during development. The use cases presented in Section 3.1.3 are also used in Chapter 4 to demonstrate the performance of ICMF against a state of the art centralized planning and scheduling tool. To setup the high level context for some of the definitions and assumptions laid out in this section, a high level block diagram of how CCBBA functions and the modifications made as part of this thesis is shown in Figure 3-1.

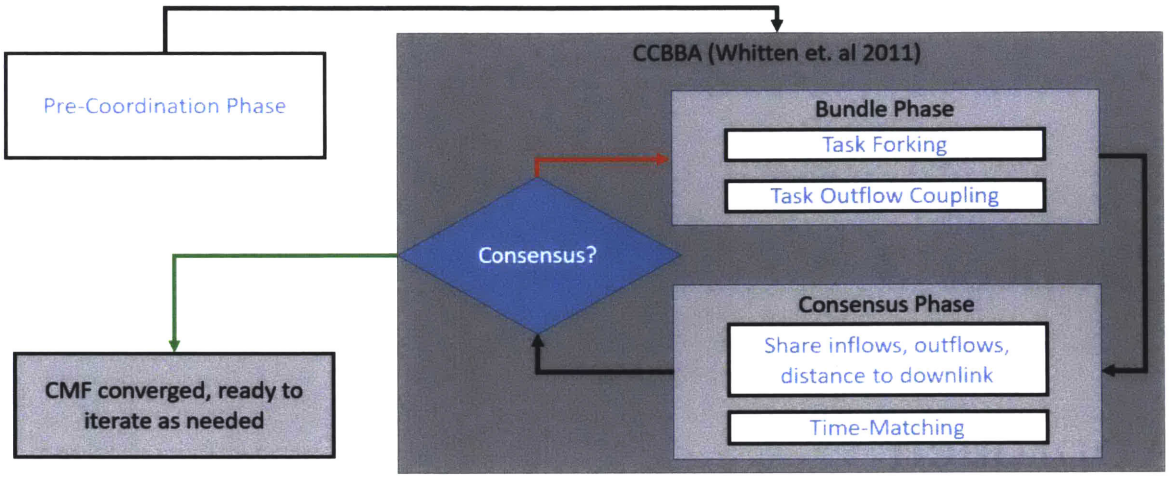


Figure 3-1: Simplified CCBBA loop with modifications for CMF shown in the white boxes with light blue text

Details on enhancements to CCBBA, which are highlighted in the white boxes with blue text in Figure 3-1, refer to Section 3.2.

3.1.1 Definitions and Assumptions

This subsection focuses on the definitions, assumptions, and a subset of the notation used in this thesis:

- *Agent* (I): an agent executing the algorithm (whether CMF or ICMF). Throughout the rest of this thesis, the terms satellite and agent will be used interchangeably. *Agent* (with italics) will be used to describe an agent object, which appears in pseudocode as I and has certain properties and methods that facilitate execution of the algorithms.

- *Task* (J): a satellite activity that involves slewing to point at a specific object and executing one of its subsystems. There are four specific types of tasks:

    1. Observation: Pointing to a ground target and using the mission payload to collect data. When this type of task is active, the state of the satellite changes with an additional 10 Watts of power draw and 50 Mbps of data coming into the satellite.

2. Downlink: Pointing to a ground station and using the radio to transmit data to the ground station. When this type of task is active, the state of the satellite changes with an additional 20 Watts of power draw and 20 Mbps of data leaving the satellite. For the downlink to be successful, this must be the only satellite attempting to downlink to a particular ground station at that time.

3. Transmission Crosslink (TX): Pointing to another satellite and using the laser communication payload to send data to the receiving satellite. When this type of task is active, the state of the satellite changes with an additional 20 Watts of power draw and 10 Mbps of data leaving the satellite. For the TX to be successful, the receiving satellite must be executing a reception crosslink at the same time with the same satellite executing the TX.

4. Reception Crosslink (RX): Pointing to another satellite and using the laser communication payload to receive data from the transmitting satellite. When this type of task is active, the state of the satellite changes with an additional 5 Watts of power draw and 10 Mbps of data entering the satellite. For the RX to be successful, the transmitting satellite must be executing a transmission crosslink at the same time as the same satellite executing the RX.

- *ExecutableTask* (J_e): a subset of time a *Task* that can be executed by the satellite. As discussed in later sections and in Appendix B.1, a *Task* object cannot be directly changed by an agent, but agents can create new *Tasks* or select *Tasks* for deletion from consideration, which is required for time matching (see Section 3.2.4). Only *ExecutableTask* objects are actually stored in a satellite's bundle. Consensus occurs on the *Tasks*, and each *ExecutableTask* only has one parent *Task*, but each *Task* can have multiple *ExecuteableTasks*. This distinction occurs because of task forking (see Section 3.2.3).

- Bundle ($\mathbf{b_i}$): A value-ordered list of *ExecutableTasks* that the satellite will

execute. Tasks in the bundle are ordered by value and are added one at time.

- Path ($\mathbf{p_i}$): A time-ordered list of tasks that the satellite will execute. This is just a book-keeping mechanism to build up the data and energy states to ensure that no state violations occur.

- Bid: the value of all *ExecutableTasks* corresponding to a single *Task* added to the bundle is recorded by the *Agent* and stored as a bid. Agents use bids to determine who should be the winner for a *Task* and achieve consensus. Bids are stored and compared across *Tasks* since all *Agents* have a consistent set of *Tasks*, while *ExecutableTasks* are created by each agent individually, see Section 3.2.3 for additional explanation. Normally, an agent cannot add a task to its bundle unless its value is higher than the highest known bid, but there will be some exceptions, as explained in Section 3.2.

- Winner: The agent with the highest bid for a *Task* is the winner. There may be multiple winners while the algorithm is running, but for consensus to be achieved, each *Task* can have at most one winner.

- Activity Timeline ($a_i(t)$): This state tracks each satellite's planned activities in the planning process. Each timestep in the planning horizon is recorded in the activity timeline with required transition times enforced in between activities. Each element of this state vector will be either: *null*, T (transition), D (downlink), O (observation), R (reception crosslink), or X (transmission crosslink).

- Energy State ($e_i(t)$): The state of each satellite's energy storage. Each satellite has a minimum storage level of 2.78 Watt-hours (Wh) and maximum storage level of 13.89 Wh. Each satellite starts the scenario with the energy state set to 12 Wh. Each satellite has a base power draw of 10 Watts, with additional power being used depending upon the task being executed. When in eclipse, there is no power generation. While in the sun, there is an assumed orbit average power (OAP) of 30 Watts. This OAP of 30 Watts is how each satellite replenishes its energy state and only occurs while not in eclipse. Satellites take energy state

into account when evaluating tasks to add to their bundle and will limit the duration of a task if it will put their energy state below the minimum.

- Data State ($e_i(t)$): The state of each satellite's data storage. Each satellite has a minimum data storage of 0 Gb and a maximum storage of 12 Gb. Each satellite starts the scenario with data state set to 0 Gb. Data State only changes as a result of tasks that the satellite executes.

- Greedy: Myopic selection of the next item, whether it is an activity to add to the bundle or a step along a route, based on that item having the highest value, without consideration for multiple items in combination.

- Multibid: A property of a task that allows multiple agents to bid on the same task and only the agent with the highest bid can execute the task. This is the general setting for CBBA tasks; however, ICMF also allows for singlebid tasks, which are tasks that only one agent can bid on and are not shared with other agents (such as individual observation windows).

- Singlebid: A property of a task where only one agent can bid on the task. This occurs in the satellite bulk data routing problem setting because access windows can occur with no overlap between different agents (e.g. for downlinks with low numbers of total satellites). Singlebid can also occur when data from all targets is desired and not just the satellite with the best observation window for each target.

- Consensus Round: A consensus round includes both the bundle phase and consensus phase of CMF. The bundle phase and consensus phase distinctions are similar to those of CCBBA, with the exceptions due to some of the enhancements in CMF described in Section 3.2. Each consensus round can have multiple communication rounds during the consensus phase. These communication rounds occur until information is propagated across the network, or consensus is achieved (all agents agree). The number of communication rounds per consensus round is limited by R_c, which is explained in Section 3.2.2.

- CMF Iteration: ICMF includes multiple iterations of the CMF with limited information to achieve consensus faster with higher value. The number of CMF iterations is bounded by $iters_{MAX}$, which is a parameter set by the user of ICMF, as described in Section 3.3.

Some assumptions are also necessary to establish the capabilities of each satellite that pertain to setting up and executing ICMF. Key assumptions include the satellite's ability to:

1. Calculate access windows to downlink locations and other satellites based on orbit propagation.

2. Perform full duplex communication to exchange bid and a subset of state information with all neighbor satellites over a low-bandwidth radio link.

3. Keep clocks well synchronized (able to sync with GPS time at least once per orbit).

4. Assess the energy and data impact of each task over the planning horizon.

5. Transition between activities within 10 seconds (configurable). This is likely too low for most activity transitions, unless the two activities have boresight lines that are less than 100° apart; however, this was used for algorithm comparison purposes because this is what the transition time was set to in the satellite model for the centralized planner [30].

Note that if ICMF is used as a planner deployed at the ground station, instead of deployed as an on-board planner, then only assumption 5 is necessary and can be changed flexibly depending on the satellite's slewing capabilities and pointing requirements.

One notation difference to highlight before going into the mathematical formulation is the use of the dictionary datatype in some of the pseudocode. This datatype holds information that can be accessed with a unique key. Using this instead of indexing the information is required because different agents will create new *Tasks*

simultaneously that have the same unique ID (used as a "key"), but may have different indices depending on the order of operations in the decentralized setting. Important examples of this datatype are the consensus dictionaries.

There are three properties that all agents need to agree on during a round of CMF to achieve consensus. Consensus is achieved for a single agent when it has communicated with all of its neighbors and there are no changes. Once an agent hears that all agents have achieved consensus, it moves on to the next phase of the algorithm. Note: instead of referring separately to each of the three dictionaries, they will be collectively referred to as the consensus dictionaries. See Appendix B.1 for more details on the notation and other properties and methods of *Agents*.

- $I.bids(J.id) \in \mathbb{R}^{N_t}$: dictionary of latest bid information available to I (*Agent*) that has the J (*Task*) bid value at the corresponding J ID. All values in the dictionary are initialized to 0. All keys, which are the unique strings to retrieve the bids, are *Task* IDs, denoted in pseudocode as *J.id*. Note: in the CBBA papers, this dictionary serves the same purpose as the "winning bids list", which is the variable: $\mathbf{y}_i(t)$ [10, 61].

- $I.winners(J.id) \in \mathbb{N}^{N_t}$: dictionary of latest winner information available to I that has the winning *Agent* ID corresponding to the J ID. All values in the dictionary are initialized to *null*. All keys are *Task* IDs, denoted in pseudocode as *J.id*. Note: the domain is $\mathbb{N}$ (all integers) because each unique ID string can be translated to a unique integer (and vice versa) based on the encoding used. Note: in the CBBA papers, this property serves the same purpose as the "winning agents list", which is the variable: $\mathbf{z}_i(t)$ [10, 61].

- $I.inds(J.id) \in \mathbb{S}^{N_t}$: dictionary of latest information about planned timesteps available to I, stored as indices in the discrete planning domain, that has the set of timestep indicies corresponding to the *Task* ID. All values in the dictionary are initialized to $\emptyset$ (the empty set). All keys are *Task* IDs, denoted in pseudocode as *J.id*. Note: the domain of $\mathbb{S}$ means that each value of the dictionary is a set.

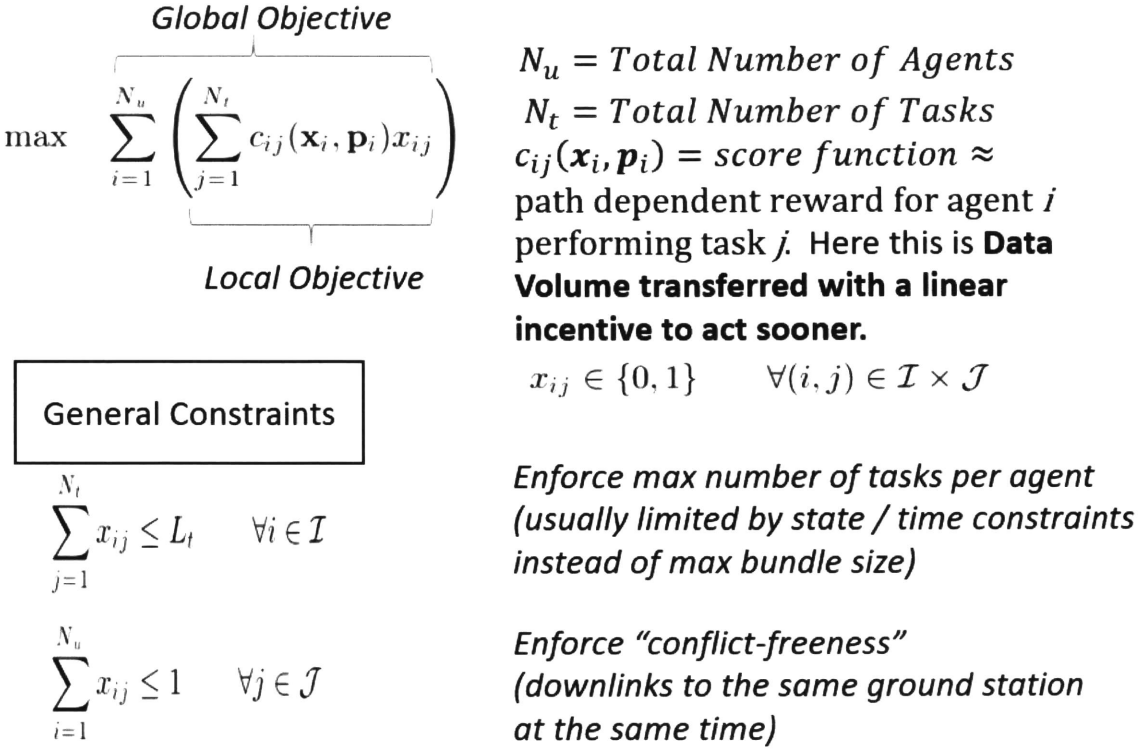


Figure 3-2: General objective function and constraints for CMF, annotated for satellite BDRP
Bolded text explains the score function used, additional details in Figures 3-3 and 3-4

3.1.2 Mathematical Formulation

With the definitions and assumptions established, the mathematical formulation of the problem will now be covered. First, the objective function is discussed, which leads to how task value is calculated. Then, the state constraints and their limits are presented.

Objective Function and Task Value

Figure 3-2 highlights the objective function and general constraints used for CMF, with some commentary specific to the satellite bulk data routing problem. This is a more specific view of the equations based on the CBBA objective function discussed in Section 2.3.3.

The global objective function is a linear function of the individual score function, $c_{ij}(\mathbf{x_i}, \mathbf{p_i})$, used for each task. For the BDRP, this thesis assumes that total data

volume delivered is the primary objective. Secondary goals include accomplishing activities earlier in the planning window and trying to preserve some data margin on each satellite. With this in mind, the task value formulation is as follows:

$$c_{ij}(\mathbf{x_i}, \mathbf{p_i}) = |DG_j| \, f_{DM}(d_i(t_s), J_e) \sum_{t=t_s}^{t_e^*} v_j(t) \tag{3.1}$$

where DG_j is the total data generated by the activity, which is negative for outflows and positive for inflows, hence the absolute value. f_{DM} is a fractional scaling factor that incentivizes inflows more when the satellite is low on data and incentivizes outflows more when the satellite is full of data. f_{DM} is calculated as follows:

$$f_{DM}(d_i(t_s), J_e) = \frac{D_{MAX} - d_i(t_s)}{D_{MAX}} \quad \text{if } J_e \text{ is an observation or RX crosslink (inflow)}$$

$$f_{DM}(d_i(t_s), J_e) = \frac{d_i(t_s) - D_{MIN}}{D_{MAX}} \quad \text{if } J_e \text{ is a downlink and TX crosslink (outflow)}$$

In Equation (3.1), $v_j(t)$ is the value of a task at a particular timestep. For all tasks used in this thesis, $v_j(t)$ is a linearly decreasing value from 1 to 0 over the time horizon, but this can be easily modified to incentivize other behaviors. Note that the calculations highlighted in this equation and equation 3.2 make the value function non-Diminishing Marginal Gain (DMG), which violates the convergence guarantees of CCBBA [61]. However, the value function is only non-DMG between inflows and outflows, *i.e.* adding an outflow makes other outflows less valuable (DMG), but makes inflows more valuable (non-DMG), so the non-DMG only occurs with a subset of task interactions. This limits the negative impact of using a non-DMG value function as described in Section 2.3.3.

The summation in Equation (3.1) occurs from t_s, which is the start time of the task, up to t_e^*, which is the new window end time based on state violations and is calculated with:

$$t_e^* = t_e - \max\left(\left\lceil \frac{D_v(d_i(t), J_e)}{\dot{D}_{J_e.type}} \right\rceil, \left\lceil \frac{E_v(e_i(t), J_e)}{\dot{E}_{J_e.type}} \right\rceil \right) \tag{3.2}$$

where t_e is the original end of the window under consideration based on the current

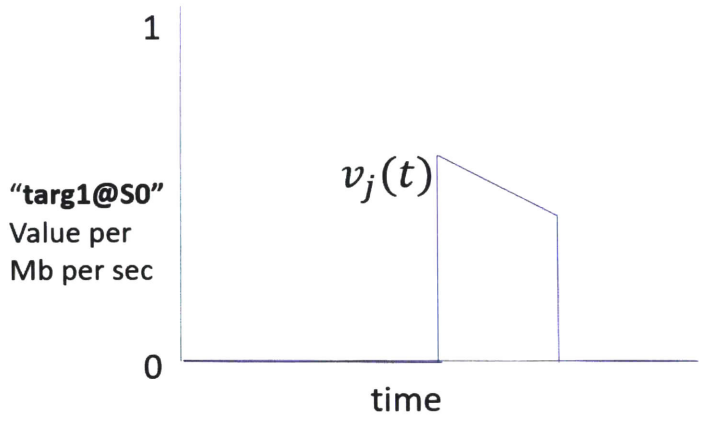


Figure 3-3: Example base task value
The notation **targ1@S0** means that this is the value of target 1 from the perspective of satellite S0. The zero portions of the value function occur when target 1 is outside the physical access window of S0. The linear decreasing function is used to motivate taking observations and downlinks sooner rather than later.

activity timeline $a_i(t)$, $E_v(e_i(t))$ is the amount of energy state violation based on the current state and execution of the full $ExecutableTask$ duration, and $D_v(d_i(t))$ is the amount of data state violation based on the current state and execution of the full $ExecutableTask$ duration. $\dot{E}_{J_e.type}$ is the energy use rate of $ExecutableTask$ J_e and $\dot{D}_{J_e.type}$ is the data use rate, which are both based on the type of $ExecutableTask$, see Table 3.1. Equation 3.2 reduces the duration of the task if the full execution will result in state violations. If this reduction puts the task below the minimum duration required for that task time, then it will receive no value. Discussed in Section 3.2.3, t_e^* limits the duration of the task that is added to the bundle now, but allows the task to be separated into different executable forks that can be added to the bundle later. The summation term in Equation (3.1) also drops any terms that are at not *null* in the activity timeline because those are times when the satellite will be already busy. This is explained in the next section and also presented as part of an example in Figures 3-3 and 3-4.

State Constraints

This section covers the state constraints and the equations used to evaluate them. There are only three states used in this problem, which were all introduced briefly in

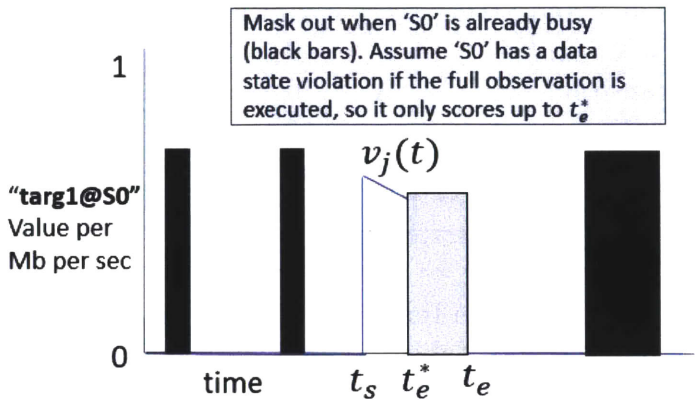


Figure 3-4: Example task value evaluated for an agent with tasks already in its bundle Task value evaluated after four tasks are already in the bundle. No value is obtained for times when the satellite is busy. Actual value calculation is based off data generated, data state at start of task ($d_i(t_s)$), and if there are any state violations from using the full duration of the task.

Section 3.1.1.

The data state constraint is formally framed as:

$$d_i^{min} \leq d_i(t) \leq d_i^{max} \quad \forall i \in \mathcal{I}, \forall t \in \mathcal{T} \tag{3.3}$$

where $d_i^{min} = 0$ Gb and $d_i^{max} = 12$ Gb for all results presented in this thesis. The energy state constraint is:

$$e_i^{min} \leq e_i(t)) \leq e_i^{max} \quad \forall i \in \mathcal{I}, \forall t \in \mathcal{T} \tag{3.4}$$

where $e_i^{min} = 2.78$ Wh and $e_i^{max} = 13.89$ Wh for all results presented in this thesis. The states are calculated based on the executable tasks in the bundle and the execution times in the path. The calculations are done with the discrete "state" equations ($\forall i \in \mathcal{I}$):

$$\begin{aligned} e_i(t+1) =& e_i(t) + \left(E_i(t)\dot{K}_{chg} + \dot{E}_{J_e.type} + \dot{E}_{base}\right)\Delta t \\ d_i(t+1) =& d_i(t) + \dot{D}_{J_e.type}\Delta t \end{aligned}$$

where $\dot{E}_{base}$ is -10 W, regardless of the activity timeline. This represents the base rate for the other subsystems on the satellite. $\dot{K}_{chg}$ is the satellite charge rate and $E_i(t)$ represents if the satellite is in the sunlight or not. $\dot{E}_{J_e.type}$ is the energy usage rate of the activity at that timestep, which is analogous to the data usage rate as a

Task Name and String	$\dot{D}_{J_e.type}$ (Mbps)	$\dot{E}_{J_e.type}$ (W)
Idle - *null*	0	0
Transition - T	0	0
Observation - O	+50	-10
TX - X	-10	-20
RX - I	+10	-5
Downlink - D	-20	-20

Table 3.1: State Impacts of Each Task Type

As explained in Section 2.1 and Table 2.3, the data and energy state values are assumed based on current or projected CubeSat capabilities.

function of activity at that timestep, $\dot{D}_{J_e.type}$, in the data state equation. Δt is the timestep between the discrete time points in the planning horizon, which is set to 10 seconds for all results in this thesis. These states are evaluated for each potential task to add to the bundle as well.

To capture the bundle and path information with variable length tasks, another state is kept, known as $a_i(t)$, which is the activity schedule at each timestep. The activity schedule is N_{steps} long with each index corresponding to the respective timestep. Only the following entries are allowed at each index:

- *null*: no activity or transition scheduled.
- T: transition occurs at this timestep. Required when tasks change nodes, based on activity transition times as specified in the satellite model.
- O: observation occurs at this timestep.
- X: transmission crosslink occurs at this timestep.
- R: reception crosslink occurs at this timestep.
- D: downlink occurs at this timestep.

The state impacts for each of the possible tasks are shown in Table 3.1:

Initial conditions for state:

$$e_i(0) = 12 \text{ Wh } \forall i \in \mathcal{I}$$

$$d_i(0) = 0 \text{ Gb } \forall i \in \mathcal{I}$$

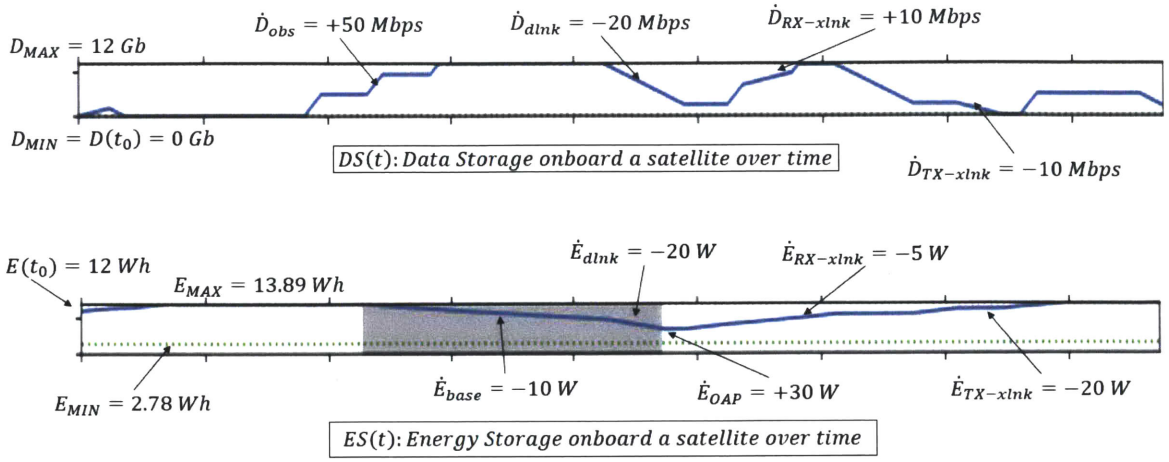


Figure 3-5: Example of a satellite's state over the planning horizon

Figure 3-5 shows an annotated example of a typical satellite's data and energy state over the course of a 105-minute planning horizon. Note that the dark grey region in the energy state plot represents the eclipse period, where $E_i(t) = 0$, otherwise $E_i(t) = 1$.

3.1.3 Test Cases

As ICMF was being developed, the main test scenario was based on a challenging planning case similar to the nominal case for the central planner used in the SPRINT project, which was the state of the art centralized planner for small satellite BDRPs described in Section 2.3.2, based on the GP-Fast planner developed by Kennedy [30]. One use case is not sufficient for testing the algorithm, especially in the performance validation phase, so additional versions were developed.

- Commonalities between all use cases:
  - 100 observation targets equally spread around the world from ± 30° latitude
  - 10 NASA NEN ground stations (see Table 2.2 for latitude and longitude locations)
  - 3-plane Walker constellation with equal spacing between all satellites in the same plane at an altitude of 600 km

- Absolute start time fixes the starting position of the constellation relative to the Sun and Earth
- Planning time horizon of 105 minutes. This time was chosen because it is close to the orbital period and aligned with the planning horizon time that the SPRINT CGP was tuned for

- Differences between use cases:
  - Inclination of the Walker constellation was either 30° or 60°. At 30° inclination, the constellation is observation dense and there is more frequent inter-plane connectivity. However, the 60° inclination constellation has more downlink opportunities per observation because a larger diversity of the NASA NEN ground stations are flown over. Both of these factors are apparent in Figures 3-6 and 3-7.
  - Number of satellites in the constellation was set to either: 30, 60, or 90. The satellites are always equally spaced, so this results in 10, 20, and 30 satellites per plane (respectively). As the number of satellites goes up, the number of unique observation and downlink windows increases linearly; however, the number of possible routes increases exponentially because the satellites come closer together, increasing the network connectivity.

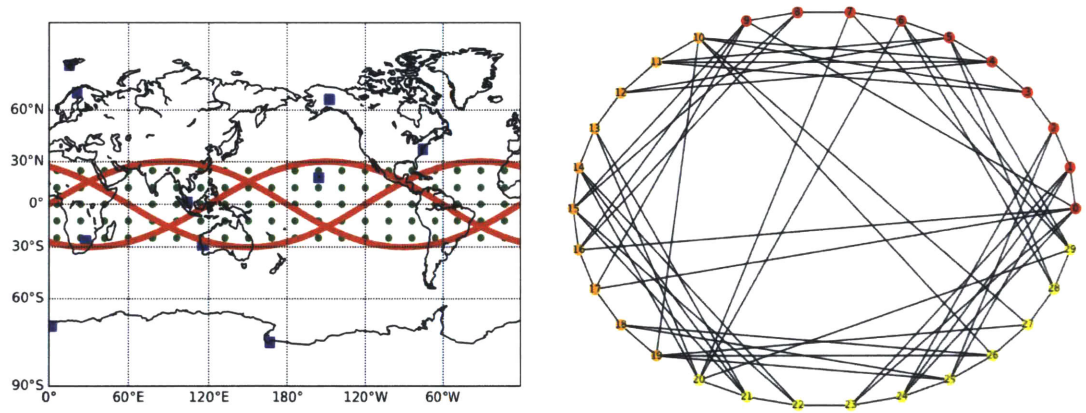
There are six unique use cases, each with different planning complexity range in terms of number of potential *Tasks* (Obs for Observations, Dlnk for Downlinks, and Xlnk for Crosslinks) to schedule, as shown in Table 3.2. The Xlnk windows are calculated based on being broken up into five minute windows.

3.2 Enhancements to CCBBA to Address the Max Flow Problem

CMF was developed with modifications to CCBBA to accommodate the nature of the max flow problem with time-gated windows of execution. This section covers those modifications by first revisiting why CCBBA formed a solid foundation for the

Use Case Name	Obs Windows	Dlnk Windows	Xlnk windows
walker30_inc30_NEN	262	56	1,080
walker60_inc30_NEN	524	111	4,392
walker90_inc30_NEN	766	164	10,248
walker30_inc60_NEN	116	58	1,272
walker60_inc60_NEN	234	115	5,208
walker90_inc60_NEN	358	175	12,864

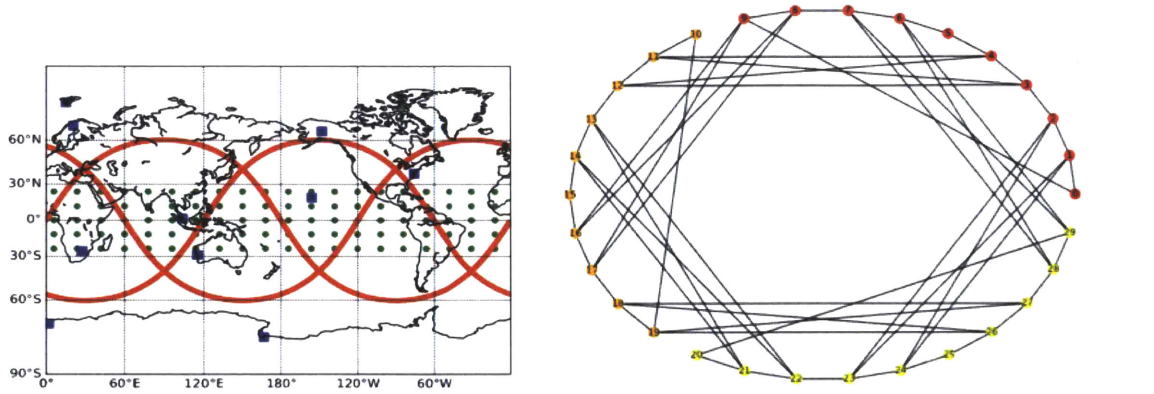
Table 3.2: Use case names and number of unique activity windows. These use cases have unique names that follow the pattern: walkerXX_incYY_NEN, where XX is the number of satellites and YY is the inclination of each orbital plane, in degrees.



(a) Miller Projection of test case with 3 planes at 30 degrees inclination

(b) t=0 satellite comunication network for walker30_inc30_NEN

Figure 3-6: Results contain 10,20, and 30 satellites per plane. In (a), green dots represent observation targets, blue squares are the ground stations, and red lines are the satellite ground tracks. This is a densely packed constellation with frequent crosslinks and saturated observation schedule.



(a) Miller Projection of test case with 3 planes at 60 degrees inclination

(b) t=0 satellite comunication network for walker30_inc60_NEN

Figure 3-7: Results contain 10,20, and 30 satellites per plane. In (a), green dots represent observation targets, blue squares are the ground stations, and red lines are the satellite ground tracks.

problem, then describing the modifications. First, the pre-coordination phase, which allows agents to create new *Tasks* that better represent the problem based on the constellation geometry, is described. Then, the modifications are explained in the context of the two main CCBBA phases: the bundle phase and the consensus phase, as shown in Figure 3-1.

3.2.1 Motivation for modifying CCBBA

As explained in Section 2.3.3, CCBBA provides the capability to address complex task allocation problems while maintaining the majority of the convergence and optimality properties of CBBA. A satellite constellation is made up of unique satellite agents that will each have different noisy situational awareness yielding different task valuations because of error in each orbit propagation and access window calculation. As mentioned in Section 2.3.3, the difference in task valuation can lead to slow convergence and excessive communication in decentralized methods that require state convergence. However, this makes CCBBA is a good starting point for creating a decentralized planning and scheduling algorithm because the convergence rate is not affected by the difference in situational awareness.

The other main motivation for modifying CCBBA instead of modifying a decentralized max flow algorithm to handle satellite constraints is based on the fact that

many of the realistic use cases may support multiple mission types, not just bulk data routing. Using CCBBA as a base for developing ICMF preserves the generality of task allocation setting, mainly: 1) handling both single bid and multibid tasks, 2) accepting different value functions and/or priorities for each target (instead of treating all data as equal in value), and 3) accommodating new tasks at any time during the plan execution. All of these factors led to the decision to enhance CCBBA to address the BDRP for small satellites, instead of modifying a max flow algorithm.

The majority of modifications to CCBBA were required because of the time-fixed nature of the tasks. The tasks are time-fixed because they are only possible during physical access windows dictated by the orbit geometry and satellite's elevation limitations for observations and downlinks. This leads to the pre-coordination phase, task forking, and exact time matching enhancements present in CMF. Other modifications were made to account for the similarities to the max flow problem, where each agent has limited data capacity and some tasks decrease on-board data, while others increase it. Taking advantage of this led to: task out-flow coupling and flow state sharing. The final reason for modification was to improve upon the convergence rate of CCBBA for this type of problem, which is where the iterative outer loop came into the algorithm and created ICMF from CMF.

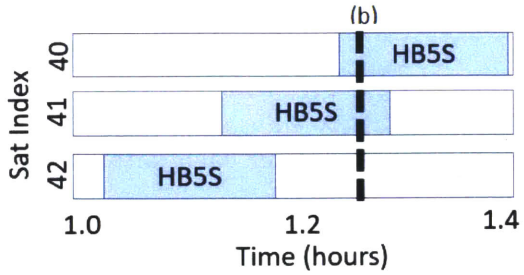
Figure 3-1 provides an overview of where the modifications occur in relation to the original execution steps of CCBBA and Table 3.3 compares the aspects of the BDRP that CBBA, CCBBA, and CMF can handle. These two graphics provide a high level map of how the algorithms are related and why the modifications were made.

3.2.2 Pre-coordination Phase

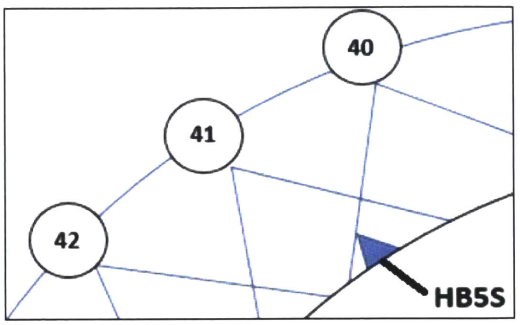
This is required because of the assumption that each satellite only knows about their own access windows to targets and ground stations. So downlink windows need to be shared to deconflict. In original CCBBA, tasks are deconflicted based only on which agent does them; however, in this case, ICMF needs to deconflict based on which agent executes a downlink task to a particular ground station at a particular time. The deconfliction moves from the binary task level (yes/no can agent X execute a

	Can Algorithm Address BDRP Aspect?		
BDRP Aspect	CBBA	CCBBA	ICMF (mechanism)
Achieve Consensus on Multibid Tasks	Yes	Yes	Yes (same as CBBA)
Mutually Dependent Crosslink Tasks	No	Yes	Yes (same as CCBBA)
Deconflict Downlink Tasks Time Overlap	No	Yes*	Yes (pre-coordination)
Revisit Remainder of a Task after Part of it is on the Bundle	No	No	Yes (task forking)
Couple another Task to Repair the State Violation from the Next Best Task	No	No	Yes (task outflow coupling)
Enforce Exact Time Match for Mutually Dependent Tasks	No	Yes*	Yes (exact time matching)
Dynamically Create new Tasks base on Flow Preferences	No	No	Yes (flow state sharing)

Table 3.3: Comparison of CBBA, CCBBA, and ICMF in addressing key aspects of the BDRP
(*) Note that CCBBA can technically handle the downlink deconfliction and crosslink time match by using "not during" and "during" temporal constraints, respectively. However, to represent all of the possible overlaps between different satellites, the temporal constraint matrix grows as $(1+2^{N_{sats}-1})$ in rows/columns because each new "not during" constraint has to split between all the previous before and after splits. For example, in the 90 satellite case for the 30° inclination constellation, 30 of the satellites have downlink windows with the same ground station at some point over the planning horizon, which results in a temporal constraint matrix of size 536,870,913 $\times$ 536,870,913. This is what led to the development of the pre-coordination phase as a way to avoid storing a matrix that large for each ground station.



(a) Subset of walker60_inc30_NEN unplanned schedule. Dashed line is time represented in (b)



(b) Downlink windows available at t=1.25 hours from use case in (a)

Figure 3-8: Example of downlink windows that need to be deconflicted Satellites 40,41, and 42 from the 60 satellite case in the 30° inclination Walker constellation are in the same plane and will all pass over ground station HB5S between the 1.0 and 1.4 hour mark in the planning horizon. This is shown in schedule form in the figure on the left. The figure on the right shows what the physical geometry looks like from a view oriented normal to the orbit plane (not to scale, approximation only).

task) to the time step level (only one agent can execute "downlink to ground station X" per timestep).

The pre-coordination phase algorithm is simple and can be explained in a few steps and accompanying graphics:

1. For the starting point, each satellite only knows their own downlink windows. Each satellite creates downlink tasks from these windows. An example of this starting point is shown in Figure 3-8.

2. The satellites communicate to all of their neighbors, sharing all of their own unique downlink windows and any other downlink windows they know about yet. A satellite sends "done" once it receives no new information from any of its neighbors and records how many rounds it took to reach "done" as R_c. All satellites stop and move to the next phase once they all have received a "done" message from every satellite. This termination condition is just a distributed flooding algorithm in a synchronous setting and will converge in $O(diam)$ rounds after the last "done" is sent [24]. Note that $R_c + 1$ is used to bound the number

of communication rounds in each consensus round during each CMF execution and in all cases tested, $R_c = \lceil diam/2 \rceil$ because in each round the satellites pull and push information from/to each of their neighbors.

3. After communication of all downlink windows is complete, each satellite runs a procedure that calculates the overlap regions for each ground station. The procedure uses powerset enumeration, which creates out the set of all subsets of time overlaps, to output unique tasks for each unique overlap set. Since all satellites have the same information, they all output the same final unique task set. An example is shown in Figure 3-9.

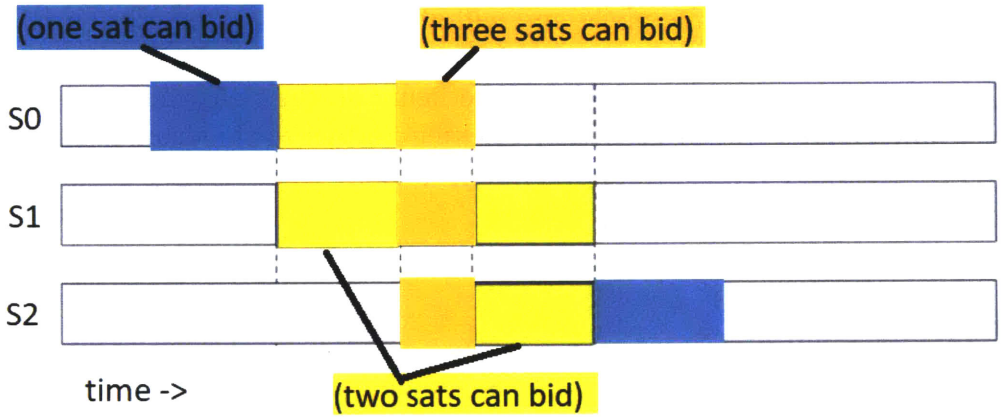


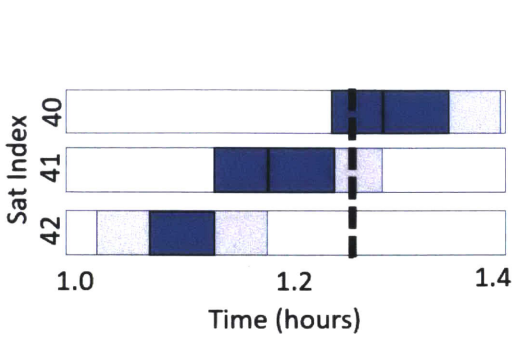
Figure 3-9: Deconflicting downlink tasks

Creating separated downlink tasks based on shared downlink window times to the same ground station

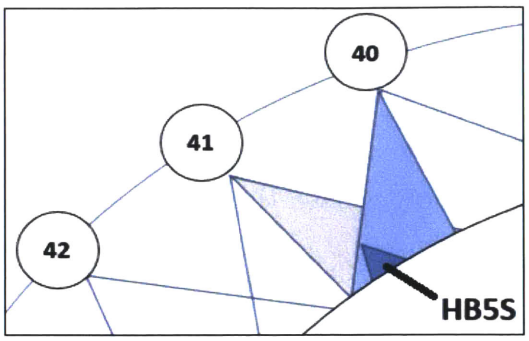
After the steps above are completed, then the rest of the algorithm starts with the first bundle phase. After planning is complete, the deconfliction of downlinks results in schedules similar to those shown in Figure 3-10 (which is a subset of the full schedule shown in Figure A-4).

3.2.3 Bundle Phase Modifications - Task Forking and Flow-Based Coupling

After completing the pre-coordination phase to create unique downlink tasks that account for time overlap at each ground station, CMF starts the bundle phase. Overall,



(a) Subset of walker60_inc30_NEN planned schedule. Dashed line is time represented in (b)



(b) Downlink windows and executions at t=1.25 hours from use case in (a)

Figure 3-10: Example of downlink schedule after deconfliction
Satellites 40,41, and 42 from the 60-satellite case at 30° inclination are in the same plane of the Walker constellation and will all pass over ground station HB5S between the 1.0 and 1.4 hour mark in the planning horizon. Figure (b) shows what the physical geometry looks when $S40$ is executing a downlink task that has a potential overlap with $S41$. The grey color indicates that $S41$ cannot downlink during this time because $S40$ won the bid for that unique task..

the bundle phase is similar to the bundle phase for CCBBA described in Section 2.3.3 where each agent individually does a greedy allocation of tasks one at a time until their bundle is full or no other task can provide value. However, some key modifications are required to address the BDRP successfully. All modifications in this section are executed by individual satellites acting alone because they take place during the bundle phase. However, the modifications can impact the convergence of the algorithm and information shared during the consensus phase and those considerations are noted throughout. Those modifications are covered in this section and the pseudocode for the entire modified bundle phase is presented in Appendix B.2.

Task Forking

Task Forking is a method used to preserve the remainder of a $Task$ for allocation at a later point in the planning process without growing the amount of information that needs to be shared between agents. By preserving the remainder of a $Task$, more total value can be achieved with the computation overhead only in the bundle phase, which is the phase that scales best with problem size. As mentioned in Section 3.1.2,

task value is based on the summation of value at each time point and the window end time, t_e, will be adjusted to t_e^* if the task will violate the data or energy state of the satellite at a time greater than t_e^*. For energy state violations, adding additional tasks will only worsen the state and revisiting the rest of a task will not provide more value; however, for data state violations, adding more tasks to the bundle could result in the rest of the task (from t_e^* to t_e) now providing value. For example:

1. Consider the observation window of target 1 by satellite 0, referred to as task $targ1\_S0$, represented by the trapezoid region in the top plot of Figure 3-11. Assume $S0$ already has four tasks on the bundle, which are represented by the black rectangles, which prevent any additional value being obtained at those times. $targ1\_S0$ is split into two regions because one task already on the bundle occurs in the middle of time window for $targ1\_S0$. $S0$ evaluates $targ1\_S0$ from t_s to t_e and finds that it will break its maximum data state if execution goes longer than t_e^* as shown in the bottom plot of Figure 3-11, so $S0$ only considers adding $targ1\_S0$ to its bundle from t_s to t_e^* and not for the full duration.

2. After going through all biddable tasks, $S0$ determines $targ1\_S0$ to be the most valuable task. $S0$ adds the $ExecutableTask$ associated with $targ1\_S0$, known as $targ1\_S0_f0$, to the bundle, only executing from times t_s to t_e^*. Then $S0$ forks $targ1\_S0_f0$ based on the total window of $targ1\_S0$ and the execution information (t_s and t_e^*) associated with $targ1\_S0_f0$. This results in the final task allocation and data state shown in Figure 3-12.

The next pass through all of the tasks during the bundle phase will include the tasks forked from $targ1\_S0_f0$. The relationship between the original and new $ExecutableTasks$ and the $Task$ is shown in Figure 3-13.

Task forking, shown in algorithm 1, occurs after a task has been added to the bundle and it's execution information has been set. Also note that algorithm 1 does not impact the consensus dictionaries directly because it only forks $ExecutableTasks$ and not $Tasks$. All of the deconfliction is done at the $Task$ level with the consensus

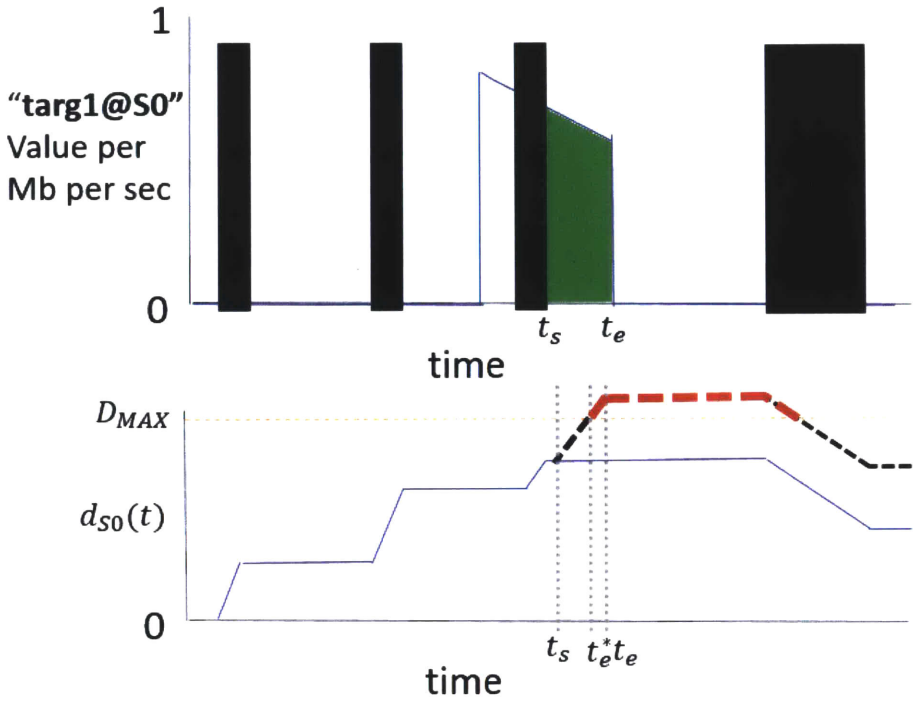


Figure 3-11: Example task that would break data state if full duration was added

dictionaries. For singlebid tasks, such as multiple contact observations, no additional coordination is required because each satellite can observe the same target without concern that it has already been observed.

However, if the task is a multibid task (such as a conflicting downlink or transmission crosslink), then an optimistic bidding strategy is used to evaluate if the set of forked tasks, which are all $ExecutableTasks$ corresponding to one $Task$, can be added to the bundle at all or not. At the end of the bundle phase, if any multibid tasks were added in an optimistic way, they are checked against the winning bids dictionary. If the accumulated value of the forked tasks does not beat the current bid, then the bundle is rebuilt with an exclusion on those tasks that did not beat the winning bid. This optimistic bidding strategy is used to avoid creating a multitude of new tasks that need to be deconflicted with other agents, which could increase the data sharing requirements and the consensus phase runtime. Refer to algorithm 6 for more details on this aspect of task forking.

Note: the method updatePostFork$_{J_e}(R_{inds}, newTasks)$ cleans up the possible ex-

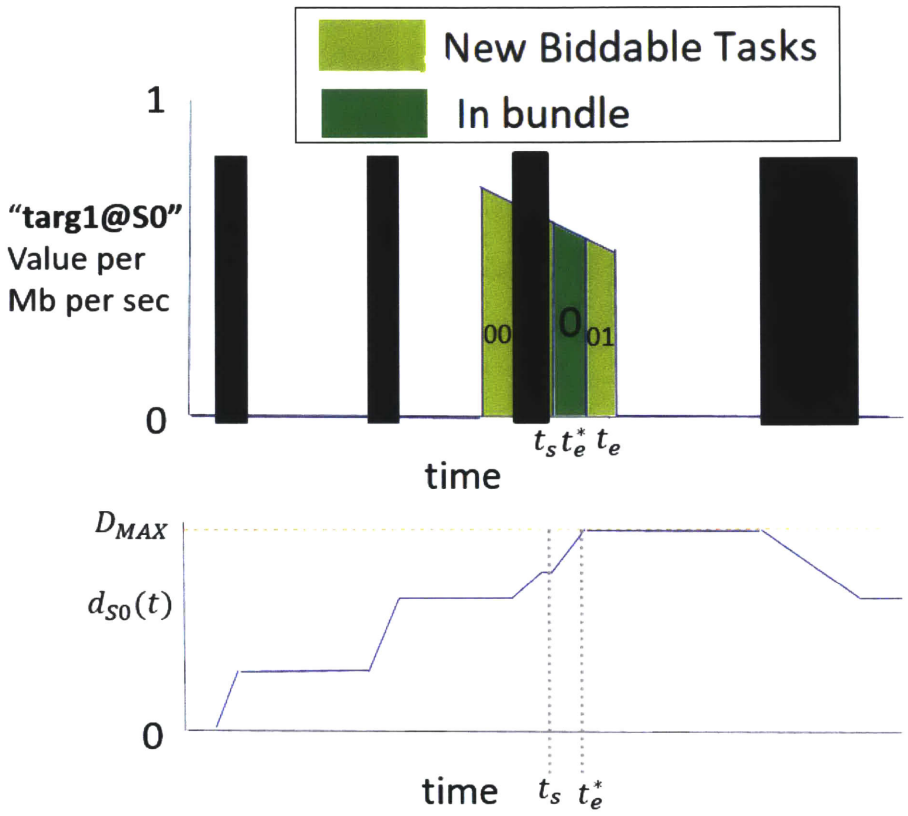


Figure 3-12: Forked tasks from adding feasible portion of a single task
Note that the light green 00 region technically includes the time up to t_s as well (black bar in the middle of the entire $Task$ window). This is because the forking process splits off new $ExecutableTasks$ to preserve the entire set of time indices present in the original $Task$. However, these indices will never be executed because that black bar represents a task already on the bundle with higher value.

Algorithm 1 Task Forking

procedure FORKTASK$_I(J_e)$
  $newTasks \leftarrow null$
  if $J_e.forkable$ **then**
    $R_{inds} \leftarrow J_e.P_{inds} \setminus J_e.E_{inds}$ ▷ Remaining t = possible t - executed t
    if $R_{inds} \neq \emptyset$ **then**
      $R_{inds} \leftarrow \text{sort}(R_{inds})$ ▷ assure in ascending timestep order
      $newTaskIndsList \leftarrow \text{split}(R_{inds})$
      $newTasks \leftarrow \text{createExecutableTasksFromInds}_I(newTaskIndsList)$
      $\text{updatePostFork}_{J_e}(R_{inds}, newTasks)$
  if $newTasks$ **then**
    $\mathcal{J}_i^{exec} \leftarrow \mathcal{J}_i^{exec} \bigcup \{newTasks\}$

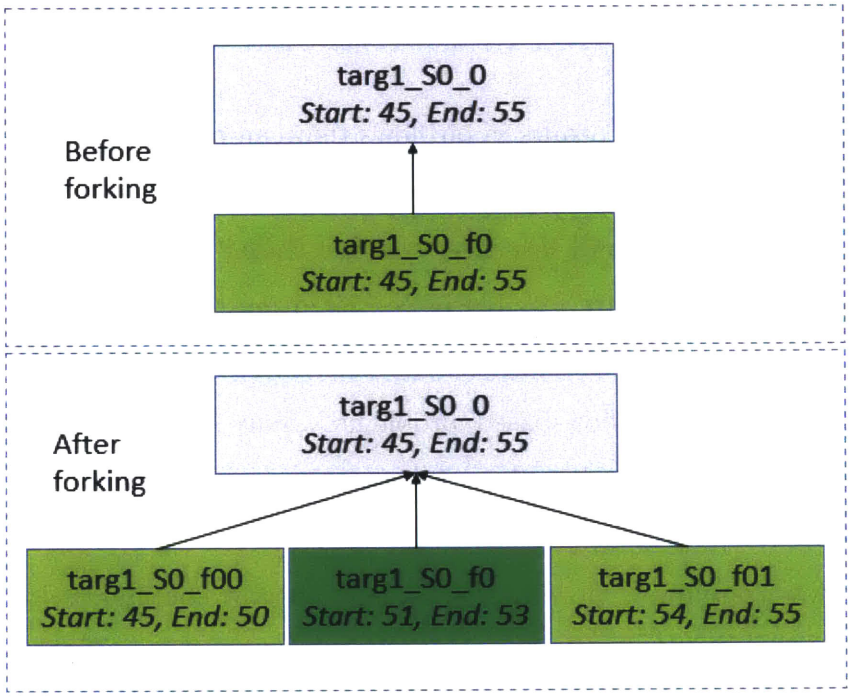


Figure 3-13: Forked *ExecutableTasks* relationship to original *Task*

Task forking is a mechanism to allow satellites to select other portions of a physical activity window (grey *Task* box) after only a subset is selected for execution (dark green *ExecutableTask* box) by allowing the remainder to made into new *ExecutableTasks* (light green boxes). Note that each *Task* originally starts with just one *ExecutableTask* covering all time points within it (top graphic "Before forking").

ecutable timesteps for the original task and adds to its forked family tree so it tracks its own forked children and parents. See Appendix B.1 for more details.

As shown in Figure 3-17, task forking significantly enhances the ability of CCBBA to handle max flow problems by allowing tasks to be segments that allow value to be obtained with interweaving of inflows and outflows, but there are cases where additional measures are required to capture more data throughput potential.

Task Outflow Coupling

The flow-nature of the BDRP results in outflows that can repair state violations that would occur in the future from inflows (and vice-versa). Task outflow coupling is used by an individual agent and leverages the flow-based nature of the problem to pair two tasks together that are feasible in combination, but infeasible when added sequentially. This can overcome some of the limitations of a sequential algorithm such as CCBBA to address flow-based problems. Since the BDRP is focused on maximizing DV downlinked, only inflows (observations and reception crosslinks) are coupled to downlinks, which is why the implementation is "task outflow coupling" and not just "flow coupling". More explicitly, task outflow coupling is invoked when an observation or reception crosslink exceeds the maximum data state at some point in the future. If a downlink can fully or partially repair that data state violation, then the original violating task is coupled to the downlink (outflow) and both are added to the bundle together in one step. Since task outflow coupling occurs in the bundle phase, there is no interaction between agents and this is a method that is used individually by each agent only.

To motivate the use of task outflow coupling, examine Figure 3-14. This figure shows a reception crosslink that would not be possible without outflow-based coupling. Reception crosslinks often benefit the most from this enhancement, which can be crucial to allow CMF to achieve throughput levels similar to that of the centralized planner. In task outflow coupling, the coupled downlink task, J_C, is added to the bundle right after the inciting inflow task, J_e, if J_e is the highest scoring task out of all available biddable tasks. This means that if J_e is removed from the bundle (e.g. if it

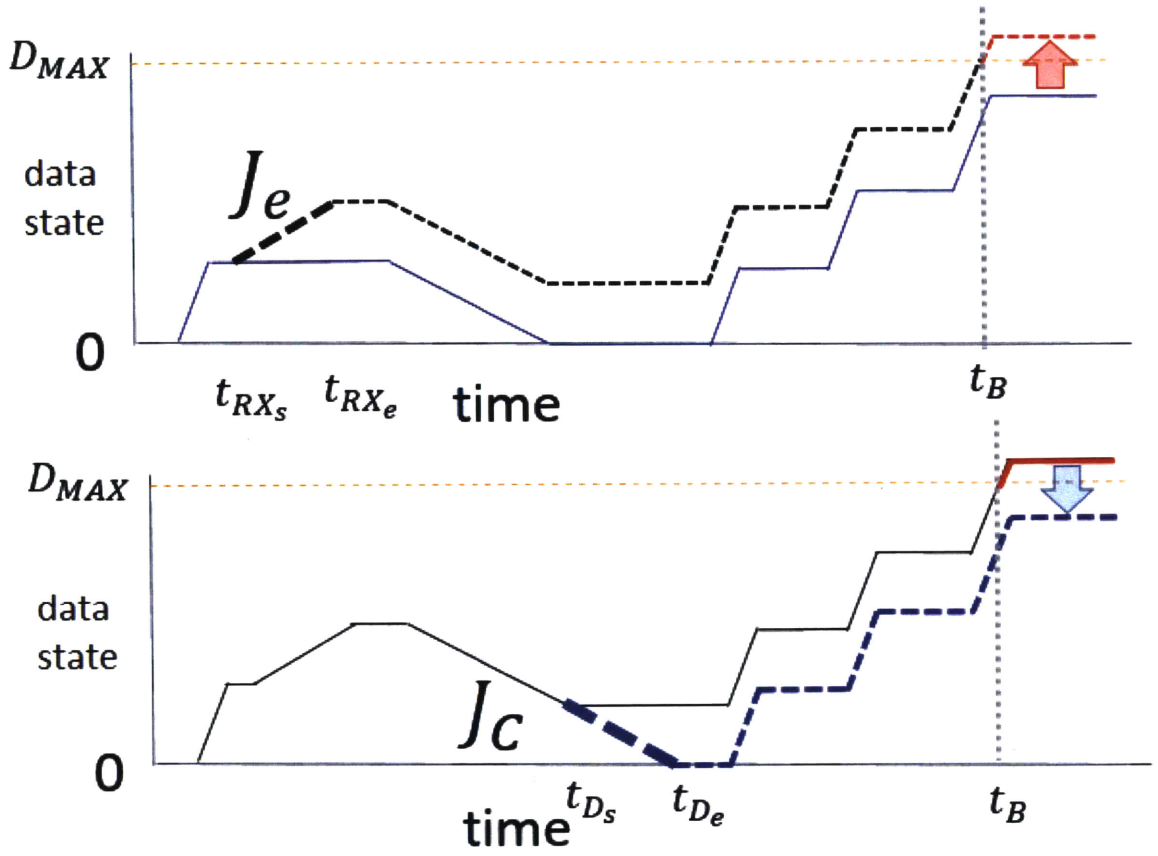


Figure 3-14: Example of Task Outflow Coupling Repairing Data State Reception Crosslink Example (J_e): The downlink (J_C) was previously forked and now the forked portion can add value when coupled to this reception crosslink because there will be data onboard at t_{D_s}. This repairs the data state violation that J_e induced at t_B.

is a reception crosslink and never matches with the mutually dependent transmission crosslink), then J_C will also be removed because it occurs later in the bundle.

The main idea behind the implementation of task outflow coupling is to allow a temporary violation of the data state constraints, save that temporary violated state, then use the same functions in a recursive approach to find the best assignment for a potential repairing downlink. If a repairing downlink exists, then it is assigned as the coupled task and if the original task is the best task, they are both added to the bundle. If no repairing downlink exists, then the original task is shortened and forked as described in Section 3.2.3. Both mechanisms can be active in the case where a partially repairing downlink is found, but the task still needs to be shortened to

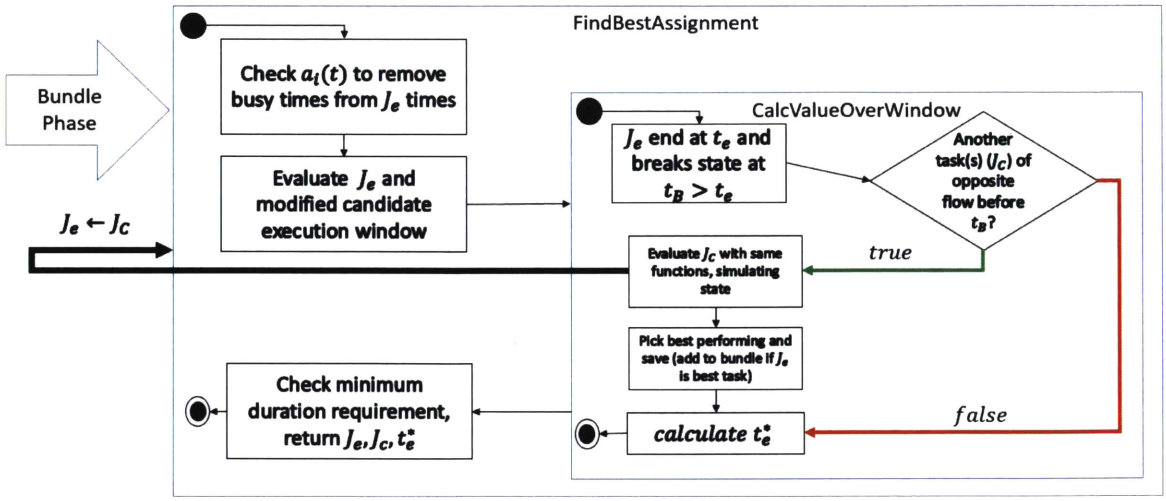


Figure 3-15: Recursion Step in Implementation of Task Outflow Coupling
Some of the details have been omitted for clarity. The solid black dot represents the function start point and the black dot with a circle around it represents function end point. See full algorithm pseudocode of the two main functions in Appendix B.2

achieve a feasible data state, even with the coupled downlink. The two main functions and the relevant recursive step are shown graphically in Figure 3-15.

Bundle Phase Enhancements Summary

This section describes an example satellite schedule and presents the results of using these modifications to show how they enhance CCBBA to better solve the BDRP.

Figure 3-16 depicts an example satellite data state and schedule with annotations explaining the impact of the modifications. A numerical evaluation of the benefit of the modifications, in terms of total runtime, data throughput, and consensus iterations, is shown in Figure 3-17. Task forking allows over 70% additional data throughput to be achieved. The sequential CBBA based algorithms, including CCBBA, without task forking, leave a lot of throughput unused because the algorithm is not able to revisit previous observation windows after more downlinks have been added to the bundle. The impact of task outflow coupling is more subtle. There is improvement in all three metrics, but the biggest effect is in reduced consensus iterations. This occurs because high value transmission and reception crosslinks can match sooner because they often occur earlier. This is especially apparent in use

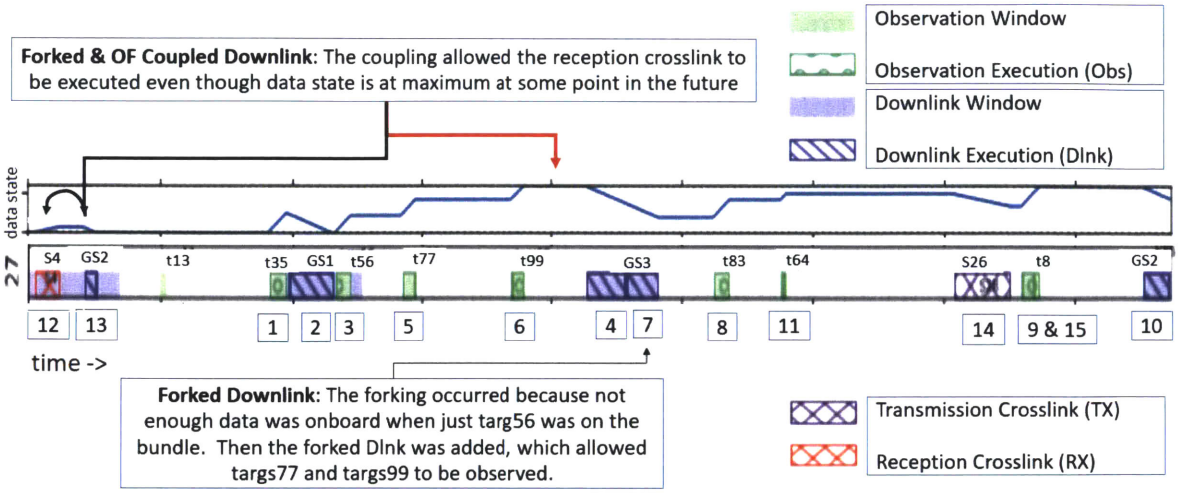


Figure 3-16: Example satellite schedule highlighting the modifications to the bundle phase of CCBBA
The first plot in this figure with linear elements represents the data state over time for satellite $S27$ in the 30 satellite case for the 30° inclination Walker constellation. The plot below that is $S27$'s activity schedule as a result of running CMF. The numbered abbreviations inside that plot represent the other physical objects that $S27$ is interacting with for each activity (either other satellites, targets, or ground stations). The numbers below the schedule represent the order that the tasks were added to the bundle, which is a decreasing value order.

cases with a lot of observations that will hit max data state at some point, where the only way to add a reception crosslink near the start is to couple it with a downlink somewhere in the start or middle of the timeline (this is also shown in Figure 3-16).

Impact of Task Forking (Observations and Downlinks only case)

	Change from baseline with coordination phase		
	With Forking		
Case (30 sats)	Runtime % change	DV % change	Iters % change
inc 30	38%	71%	**0%**
inc 60	7%	12%	**0%**

Forking improves DV throughput because the unused portion of a task can now be allocated later in the bundle

Impact of Task Outflow Coupling (all Task types)

	Change from baseline with coordination phase		
	With Forking and OF Coupling		
Case (30 sats)	Runtime % change	DV % change	Iters % change
inc 30	-12%	1%	-23%
inc 60	-13%	2%	-13%

OF coupling improves runtime and DV throughput because crosslinks that are more likely to couple are chosen sooner

Figure 3-17: Annotated tables highlighting the impact of the bundle phase modifications to the 30 satellite test cases.

3.2.4 Consensus Phase Modification - Exact Time Matching and Flow State Sharing

The bundle phase modifications help achieve higher performance on the BDRP; however, they do not address all aspects of the problem to create a physically realistic schedule. The first reason is that exact time matching between crosslinks is not enforced, only matching between mutually dependent tasks with the same mechanisms as CCBBA is used. While this ensures that a TX crosslink will occur around the same time an RX crosslink, the two satellites could choose only a small portion that overlaps in exact time. To fix this, a mechanism to enforce exact time matching by removing non-matching tasks and creating tasks more likely to exactly match based on current plan information was developed.

The other main modification was needed because using fixed crosslink windows led to excessive combinations of non-matching mutually dependent crosslinks. This caused slow convergence and a large number consensus rounds, making physical deployment of the algorithm less feasible. To address this, a flow direction preference state, or "flow state", is calculated for each agent at each timestep and then shared with other agents during consensus rounds. This flow state is used to match inflows with outflows and create fewer total crosslink tasks, while still keeping the most valuable crosslink possibilities.

Exact Time Matching

While crosslinks may match at the task level, there is no mechanism in CCBBA to enforce an exact match for each timestep (except for the temporal constraint matrix, which was deemed to grow too large for this task, see Table 3.3). For example, crosslink task "TX:S0→S1_10-20" and "RX:S0→S1_10-20" represent a crosslink from satellite S0 and S1 from time indices 10 to 20. Both satellites have to bid on the respective task, but they may execute different portion of the task (e.g., S0 plans to execute "TX:S0→S1_10-20" from 10 to 15 (before running into a potential data state violation) while S1 executes "RX:S0→S1_10-20" over the entire duration from 10 to 20). If this were to go uncorrected, then satellite S1 would think it has 5 steps worth

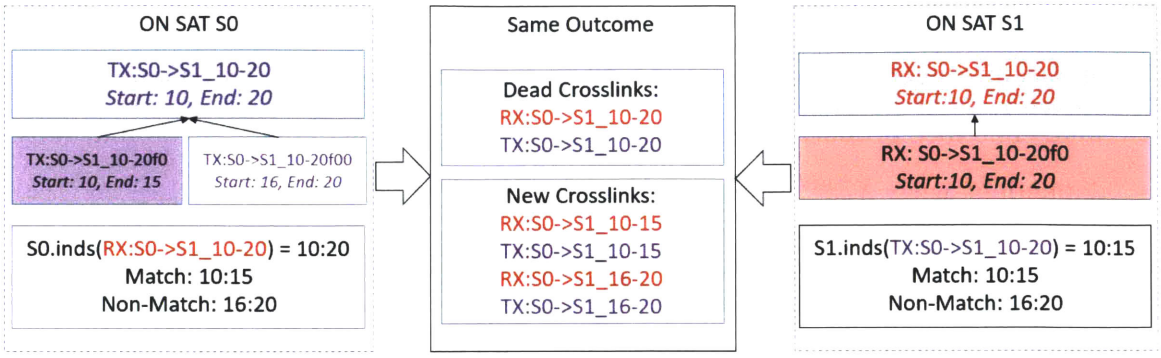


Figure 3-18: Example of Time Matching Algorithm Creating New Crosslinks
Red signifies a reception crosslink and purple is used for transmission crosslinks. Filled boxes signify the $ExecutableTask$ that is actually on the bundle. Even though the two satellites perform the operation independently at the end of the consensus phase, they come to the same outcome (box in the middle of the figure) because they have shared information.

of reception crosslink data that was never actually delivered yielding an incorrect plan. To fix this, satellites also share their planned execution times for all crosslinks with each other during the consensus phase. At the end of the consensus phase, any satellite that has met the mutual dependence requirement for a crosslink task then checks the execution times. If the execution times do not match, then new tasks are created at both agents based off the time disagreement. This example is shown in Figure 3-18.

In Figure 3-18 "Dead Crosslinks" means that they are no longer used in the algorithm. These crosslinks have all the $ExecutableTasks$ removed from the bundle and are communicated to other satellites during the consensus rounds to remove them from their respective consensus dictionaries. "New Crosslinks""are added to the consensus dictionaries and have corresponding $ExecutableTasks$ created. The time matching procedure is shown in algorithm 2 and executes at the end of each consensus round. During each consensus round, each satellite also shared their list of "All Dead Crosslinks" with their neighbors so that they can remove them from their consensus dictionaries; this step is not shown in algorithm 2. Some notation is introduced below to simplify some of the book-keeping required.

- X_i^{valid}: this is a specific type of $Task$ (J), which has the following properties:

type of $xlnk$ (TX or RX), one or more of its $ExecutableTasks$ is on the bundle of satellite agent with index i, and has its mutually dependent $Task$ met as well. The last property means that the bid value of the $Task$ that is mutually dependent with X_i^{valid} is greater than 0, $I.bids(X_i^{valid}.MD.id) > 0$.

- $\mathcal{X}_i^{valid}$: this is the set of all valid crosslinks on the bundle of agent with index i.
- For other properties, methods, or simple functions not yet defined, refer to Appendix B since they are used in multiple pseudocode blocks.

Algorithm 2 Crosslink Time Matching

procedure MATCHTIME$_I(\mathcal{X}_i^{valid})$
  $dead_{xlnks} \leftarrow \emptyset$
  $new_{xlnks} \leftarrow \emptyset$
  for X_i^{valid} **in** $\mathcal{X}_i^{valid}$ **do**
    $t_x \leftarrow I.inds(X_i^{valid}.id)$ ▷ exec timesteps for $xlnk$
    $t_y \leftarrow I.inds(X_i^{valid}.MD.id)$ ▷ exec steps for mutually dependent $xlnk$
    if $t_x \neq t_y$ **then**
      $dead_{xlnks} \leftarrow dead_{xlnks} \bigcup \{X_i^{valid}, X_i^{valid}.MD\}$
      $match \leftarrow t_x \bigcap t_y$
      $all \leftarrow X_i^{valid}.P_{inds}$ ▷ all possible timesteps for this $xlnk$
      $nonMatch \leftarrow all \setminus match$ ▷ set difference operation
      $nonMatches \leftarrow$ split$(nonMatch)$ ▷ sets of continuous timesteps
      $newIndsSets \leftarrow \{match, nonMatches\}$ ▷ set of separated sets
      $new_{xlnks} \leftarrow new_{xlnks} \bigcup$ createNewXlnks$(newIndsSets, X_i^{valid})$
  removeAllExecFromBundle$_I(dead_{xlnks})$
  initNewXlnks$_I(new_{xlnks})$
  $I.AllDead_{xlnks} \leftarrow I.AllDead_{xlnks} \bigcup dead_{xlnks}$

In most cases, the percentage of crosslinks that exactly match in time (as a percentage of the crosslinks that match at the $Task$ level) is above 60% after the first round with crosslinks present, with an example shown in Figure 3-23. This is because ICMF only creates crosslinks based on matching inflow with outflow states at each timestep. This flow matching yields crosslinks that are shorter duration, while still having potential value. The user of ICMF can tune the minimum duration of all activities to ensure windows are not too short. This reduces the amount of crosslink removal and creation from the time matching procedure because a lot of crosslinks are

set for full duration. Task forking also helps reduce non-time matches because after one task has matched at the *Task* level, more forked *ExecutableTasks* will fill in for that same *Task*, resulting in higher likelihood of both *Tasks* being full duration since they can revisit the original duration until it has been fully allocated to the activity.

Flow State Sharing

With the bundle phase modifications and crosslink time matching, all of the infrastructure is in place to start testing CMF performance. Performance with just these modifications is reasonable and comes within 5% for data throughput of the SPRINT CGP. However, the runtime is only slightly better and the data throughput as a function of consensus rounds is flat. While this shows the feasibility of the method, it also reveals that using fixed duration windows for the crosslink tasks results in an inefficient information sharing approach. In the initial approach, crosslink *Tasks* are created by breaking up physically continuous crosslink windows between satellites into 5 minute chunks and calling each time chunk a separate *Task*. This preserves all possible physical crosslink opportunities, but also creates a large number of crosslinks and presents an arbitrary time barrier between physically continuous windows. The performance limitations of this approach are highlighted for one example in Figure 3-19.

After reviewing these initial results, a new approach was developed based on a flow preference state, either "inflow" or "outflow", that is calculated for each timestep at the end of each bundle phase. An "inflow" preference means that the satellite can take reception crosslinks at that time and and "outflow" preference means the satellite can take a transmission crosslink at that time. These preferences are only a function of the agent's bundle and accumulated knowledge about other satellites up to that point. Agents share their flow direction preference with their neighbor during each consensus round and they keep a dictionary of all other agents "inflows" and "outflows". This is a key point because the flow state is a property of that agent which other agents use to determine if they should create crosslinks that involve that agent. TX crosslinks are created for each timestep where an agent's set of outflows

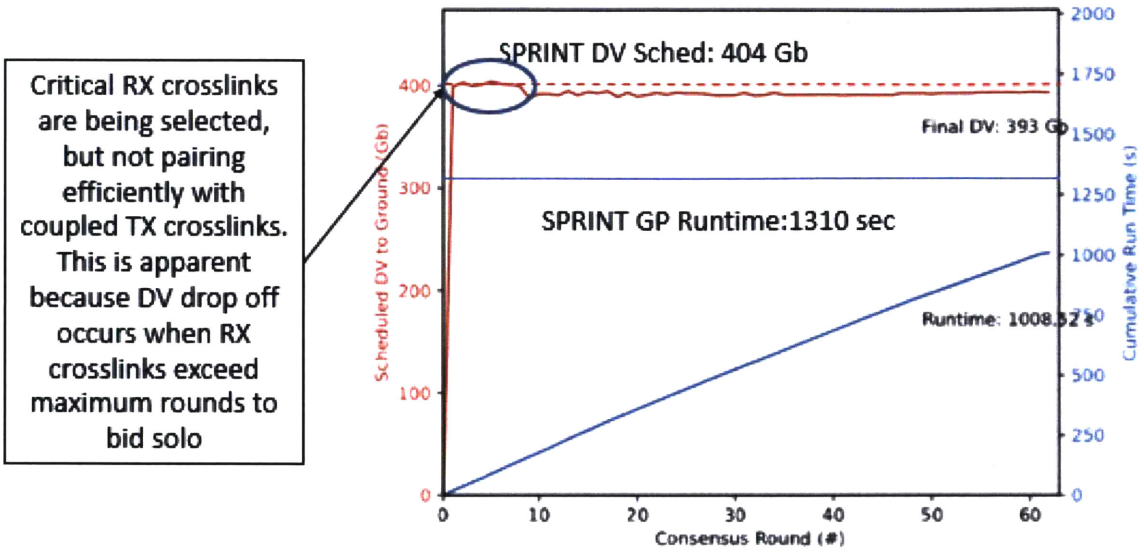


Figure 3-19: Performance of fixed crosslink windows
Use Case: 30 satellites in 30° inclination walker. With fixed crosslink windows, CMF achieves 23% faster runtime, but with 3% less DV throughput than the SPRINT CGP.

matches with a neighbor's set of inflows. That neighbor will create the mutually dependent RX crosslink at the same timesteps by matching their own inflows with the original agent's outflows. This approach is flexible in that the user can program a function to calculate the direction preference at each time step. The flow direction preference should be related to the global objective function for best performance.

The flow direction approach is summarized with an example shown in Figure 3-20.

The particular function used to determine flow preference state and how it fits into the rest of CMF is shown in algorithm 3. This is run at the end of each bundle phase. A critical part of this function is the use of the "distance to downlink for agent I" or $dist_I^{dlnk}(t)$. This represents the network distance, measured in hops, that a satellite agent is from a downlink as a function of time. If a satellite has an open downlink (*i.e.* a downlink window that is not fully executed) at some point t_{od} then all times $t < t_{od}$ will have $dist_I^{dlnk}(t) = 0$ since that satellite can downlink new data itself. Otherwise, $dist_I^{dlnk}(t)$ is based on the minimum of all neighbors $dist_I^{dlnk}(t)$. Mathematically, it is calculated as follows:

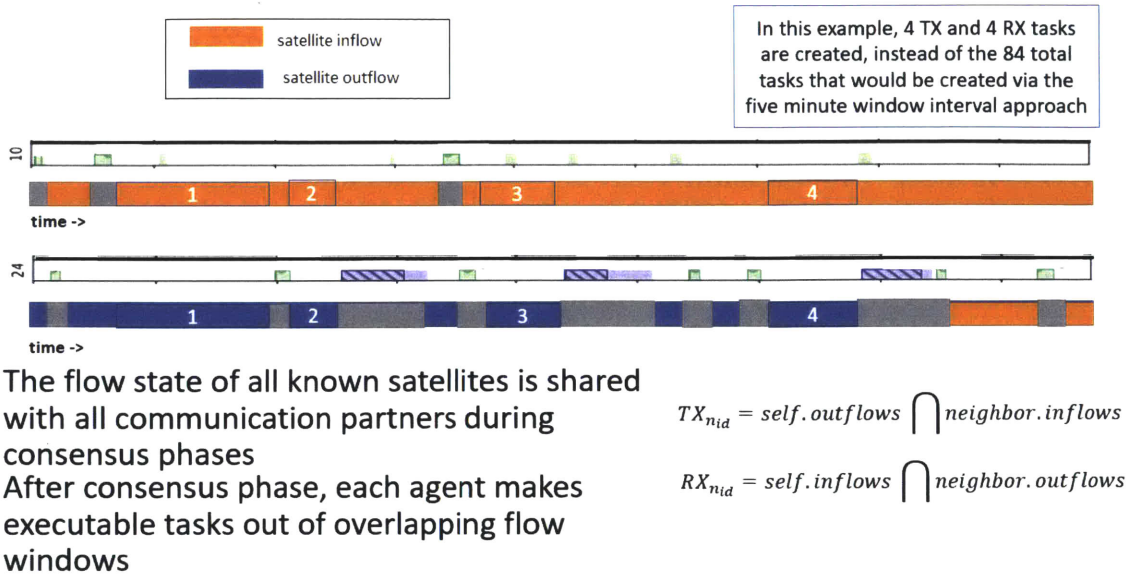


Figure 3-20: Example of flow direction state
Chart describing flow direction windows in one example schedule after the first bundle phase. Satellites calculate flow preference state for each timestep in the planning horizon. The bars underneath each schedule represent the flow preference state: blue is an inflow preference, orange is an outflow preference, and gray is *null* (busy at that time).

$$minDist^{dlnk}_{neigh}(t) = \min_{\forall i \in \mathcal{I}}(dist^{dlnk}_{i}(t)) + 1 \quad \forall t \in \mathcal{T} \tag{3.5}$$

$$dist^{dlnk}_{I}(t) = \min(minDist^{dlnk}_{neigh}(t), dist^{dlnk}_{I}(t)) \quad \forall t \in \mathcal{T} \tag{3.6}$$

During the first bundle phase (before any information sharing about planned bundles), $dist^{dlnk}_{I}(t) = \text{inf} \quad \forall t > t_{od}$, where t_{od} is the timepoint of the last open downlink in its bundle. If there are no open downlinks, then $dist^{dlnk}_{I}(t) = \text{inf} \quad \forall t \in \mathcal{T}$. This functionality is referenced in the pseudocode of algorithm 3 by the function updateDistanceToDlnk$_I$(). The flow direction state, $I.fd(t)$ is stored as an array of strings, where each entry will either be 'I', 'O', or *null*, which stands for "inflow", "outflow", or "no flow", respectively. "No flow" occurs when the satellite is already busy at that time step with another task or transition.

Note that f_{ds}, the fraction of data storage heuristic, controls the balance between inflow vs. outflow at time t when $dist^{dlnk}_{I} \neq 0$. This can be used to tune the balance between data margin preservation, performance, and consensus rounds. Values closer

Algorithm 3 Flow Direction Preference Calculation

procedure CALCFLOWDIRECTION$_I(f_{ds})$
  $dist_I^{dlnk}(t) \leftarrow$ updateDistanceToDlnk$_I()$
  for t **in** $\mathcal{T}$ **do**
    if $dist_I^{dlnk}(t) = 0$ **then**
      $I.fd(t) \leftarrow$ 'I'
    else ▷ no open downlinks remaining for this agent at times $> t$
      if $d_i(t) < f_{ds}(D_{MAX} - D_{MIN})$ **then**
        $I.fd(t) \leftarrow$ 'I'
      else
        $I.fd(t) \leftarrow$ 'O'
    if $a_i(t) \neq null$ **then**
      $I.fd(t) \leftarrow null$ ▷ busy during this time, overwrite with no flow

to f_{ds}=0.5 will create more total crosslinks, but could lead to more consensus rounds and less data margin. For the results shown for the rest of this thesis, $f_{ds} = 0.1$, because a lower value biases towards outflow when there are no open downlinks available for that agent. The outflow bias increases the likelihood that the satellites that still have open downlinks will create matching transmission and reception crosslink tasks since there are more total outflows to match with. The bias is there because the objective function focuses on total data throughput and trying to allocate all of the open downlinks will maximize data throughput. This functionality could be improved by calculating the average distance to downlink at each timestep and then using that information as well to influence the flow preference state. In its current implementation, the performance gain is shown in Figure 3-21.

With the flow preference based crosslink creation approach, the runtime and number of consensus rounds required improve drastically as shown in the annotated box in the lower left of Figure 3-21, with only a small additional drop off in total data volume throughput performance as compared with using fixed crosslink windows. Lowering the number of consenus rounds is a key requirement to make this approach physically realizable on small satellites, so this was determined to be the better approach to creating crosslink windows. However, the performance as a function of consensus rounds was still flat and the last 50% of consensus rounds had little effective information sharing (*i.e.* information which impacted the actual final schedules vs. information

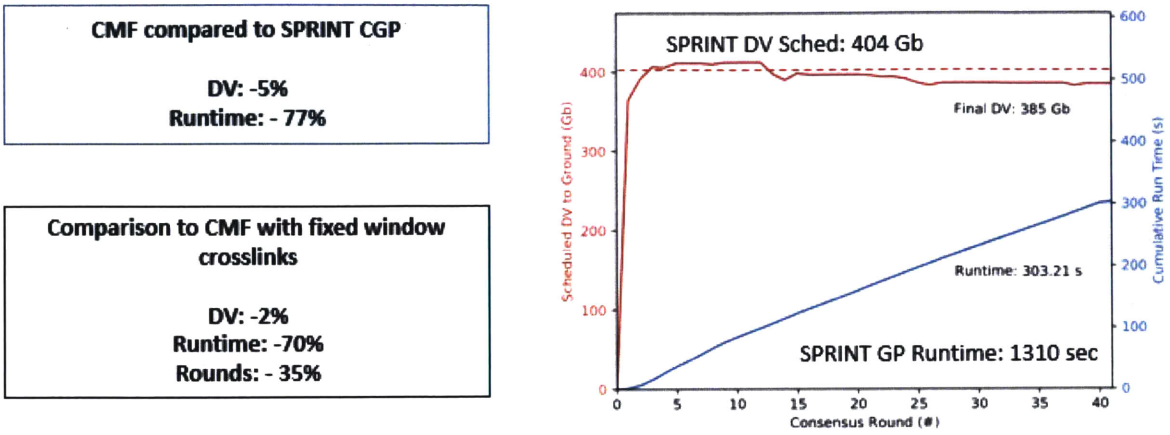


Figure 3-21: Performance of flow based crosslink windows
Data throughput performance drops slightly, but runtime is much better and consensus rounds improves. However, still a long flatline of performance as the final crosslinks are being agreed too.

that is just repeated), so an approach which only created new crosslinks outside of CMF and then iteratively ran CMF to improve performance was developed. This approach is the final overall algorithm, known as ICMF.

3.3 Iterative CCBBA for Max Flow problems (ICMF)

While CMF, with modifications to both the bundle and consensus phases of CCBBA, showed promise in being able to address the BDRP in a decentralized way, still improvements needed to be made in the convergence rate.

3.3.1 Improving Convergence of Valuable Reception Crosslinks

The main idea behind ICMF was to reduce total crosslinks created while keeping the most valuable ones. By focusing on reception crosslinks, and only creating new transmission crosslinks that paired with value-adding reception crosslinks, the algorithm could converge faster while keeping similar levels of total data throughput. The iterative approach comes from running CMF in an inner loop. Value-adding RX crosslinks are found by all satellites running a single "virtual" bundle phase with all potential RX crosslinks based on flow preference state at the end of a CMF iteration. This results in only a subset number of crosslinks used in each CMF run, except for new crosslinks created for time matching purposes as described in Section 3.2.4.

However, the new crosslink creation for time matching does not increase convergence time significantly since the new crosslink times are always a subset of timesteps of the original crosslink times. For the first iteration when CMF is called, no crosslinks are available, so only observations and downlinks are added to the bundle. Coordination is still required for downlink deconfliction in this first round.

Before continuing on, reviewing terminology to keep the various types of phases, rounds, and iterations consistent is helpful. Starting from the outer most loop:

- ICMF Iteration: This is the outer loop of ICMF, as shown in Figure 3-22. One iteration calls CMF once and either creates new crosslinks and re-enters to start the next iteration or terminates. Only on the initial iteration is the pre-coordination phase run to share all downlink windows for deconfliction, as described in Section 3.2.2.

- CMF Consensus Round: Inside CMF, there are consensus rounds, which include all agents completing the bundle phase and consensus phase of the algorithm.

- CMF Bundle Phase: In this phase of a consensus round, each agent allocates tasks in sequential greedy manner with the enhancements described in Section 3.2.3.

- CMF Consensus Phase: In this phase of a consensus round, each agent executes R_c communication rounds to fully propagate information across the network and performs crosslink time matching as described in Section 3.2.4.

Subsequent rounds of CMF occur after the execution diagram shown in Figure 3-22 loops back to the CMF block. ICMF terminates when there are no new crosslinks created (termination condition on the left side of Figure 3-22), or when a counter is reached for the maximum number of CMF iterations (termination condition on the right side of Figure 3-22). A more in-depth discussion on a sensible value of the maximum CMF iterations counter, $iters_{MAX}$ is in Section 4.2.

The high level pseudocode for ICMF is shown in algorithm 4. Note that this is written from the point of view of a "manager" process that could run the algorithm

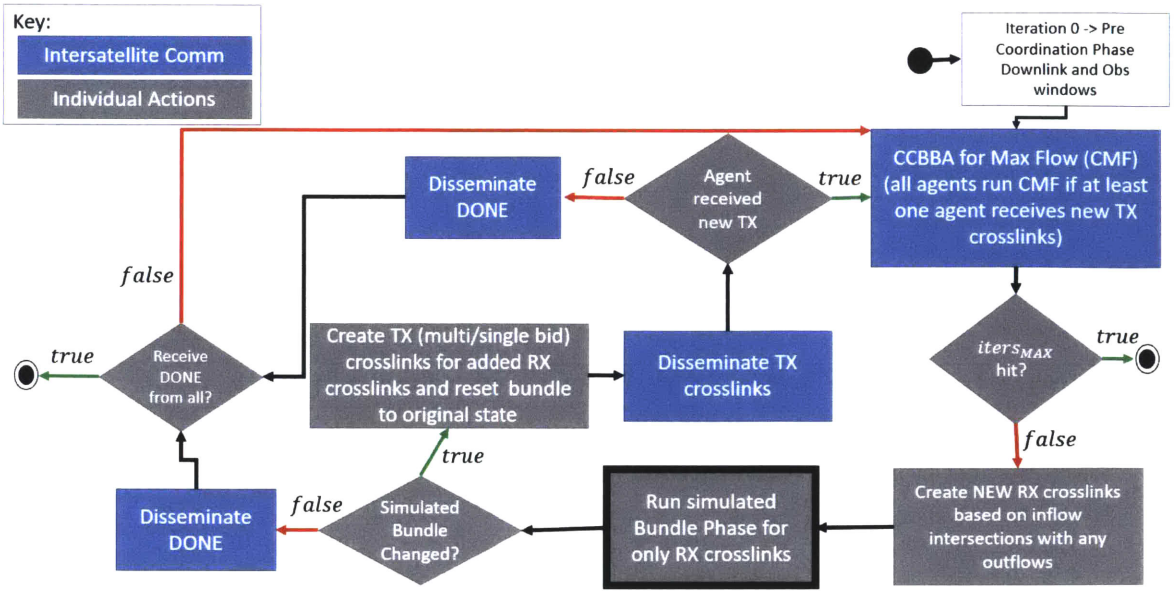


Figure 3-22: Diagram showing the outer loop of ICMF, which iterates rounds of CMF. The box with the thick outline represents a key step in ICMF that limits the number of crosslinks considered to only value-adding reception crosslinks.

on a single CPU; however, the actual implementation is intended for a decentralized environment. This pseudocode is just intended to give a sense of the overall flow of the algorithm and not describe the details, see Appendix B.2 for more details and descriptions of the functions used. The two input parameters are $iters_{MAX} \in \mathrm{N}$ and $TX_{multibid} \in \{true, false\}$. $iters_{MAX}$ is a cap on the number of CMF iterations that occur. This parameter is used to help stop the algorithm once the majority of value adding crosslinks have been found and scheduled. $TX_{multibid}$ is a boolean flag that is left as an algorithm parameter to let the user select for creating multibid transmission crosslinks or not. A parameter study that focuses on the algorithm runtime and performance as a function of these two parameters was conducted and is presented in Section 4.2.

The results of using ICMF are presented in Chapter 4, but one example with 90 satellites is shown in Figure 3-23. Figure 3-23 is intended to display some of the key properties of ICMF: the majority of consensus rounds are sharing important information which is either helping improve exact time matching, crosslink coupling, or data throughput. Note that the temporary drops in performance occur when

Algorithm 4 Iterative CCBBA for Maximum Flow problems (ICMF)

1: **procedure** ICMF($iters_{MAX}, TX_{multibid}$)
2: propagateOrbitsAndCalculateWindows()
3: deconflictDownlinks()
4: $iters \leftarrow 0$
5: $newXlnks \leftarrow true$
6: $newXlnksSet \leftarrow \emptyset$
7: **while** $newXlnks$ **do**
8: $consensus \leftarrow false$
9: **while not** $consensus$ **do**
10: shareNewXlnks($newXlnksSet$)
11: cmfBundlePhaseAll()
12: $consensus \leftarrow$ cmfConsensusPhaseAll()
13: $iters \leftarrow iters + 1$
14: **if** $iters \geq iters_{MAX}$ **then**
15: **return** "Complete: Reached CMF Iterations Timeout"
16: $AllXlnks_{RX} \leftarrow$ createRXFromFlowState()
17: $VA_{RX} \leftarrow$ tempBundlePhaseAll($AllXlnks_{RX}$)
18: $mutuallyDependentXlnks_{TX} \leftarrow$ createTXFromRX($VA_{RX}, TX_{multibid}$)
19: $newXlnksSet \leftarrow AllXlnks_{RX} \bigcup mutuallyDependentXlnks_{TX}$
20: **if** $newXlnksSet \neq \emptyset$ **then**
21: $newXlnks \leftarrow true$
22: **else**
23: $newXlnks \leftarrow false$
24: **return** "Complete: No more valuable RX Crosslinks"

crosslinks are removed from the bundle because they hit the timeout parameter limits, ω_{ij}^{solo} and ν_{ij}, that enforce mutual dependence between TX and RX crosslinks.

3.3.2 Considerations for Making Transmission Crosslinks Multi-bid

In Section 3.3.1, the ICMF algorithm parameter $TX_{multibid}$ was introduced. The difference between the two settings of this parameter are shown graphically in Figure 3-24 and are explained in detail as follows:

- $TX_{multibid} = true$: will cause ICMF to create new TX crosslinks that are split up based on the unique overlaps of all agents that have an outflow state for a single agent's inflow state. This is similar to how downlinks are deconflicted based on overlap in physical windows to the same ground station, only now the reception crosslink agent is the ground station and the time periods that the reception crosslink agent is in an inflow state is the total physical access window for the ground station. For downlinks the number of potential overlaps is usually limited to satellites in the same plane, so the growth of overlaps is manageable. This is not the case for crosslinks and the growth in potential overlaps increases dramatically as satellite plane density increases because both in-plane and crossplane crosslink opportunities grow. The main advantage of this setting is that the RX crosslink never has to choose different potential TX crosslinks to match with. Every timestep of RX crosslink time only matches with one TX crosslink, but that TX may have mutiple bidders.

- $TX_{multibid} = false$: will cause ICMF agents to create separate new TX crosslinks for every neighbor agent that has a matching outflow with its own inflow states. rRecall from Figure 3-20 that TX with a given neighbor is created when $TX_{n_{id}} = self.outflows \bigcap neighbor.outflows$. This results in less total crosslinks because they do not need to be separated based on different overlaps, but it also has the potential for lost RX crosslink opportunities if the RX and TX crosslinks don't line up before the timeouts are reached on each task. This

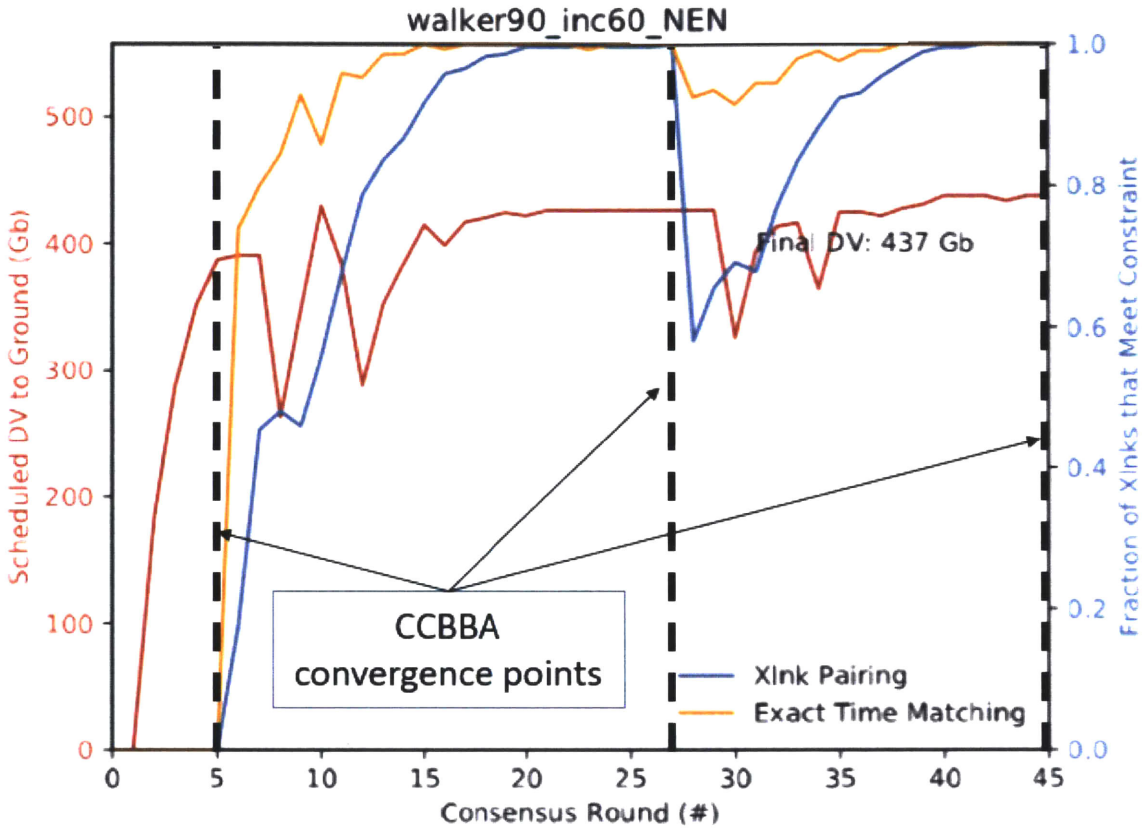


Figure 3-23: Example of ICMF convergence with 90 satellites
This example has three CMF iterations, indicated by the dotted vertical lines. On the first iteration, only observations and downlinks are present, so convergence completes in 5 rounds. Then new crosslinks are created following the approach outlined in this section. After the 2nd iteration of CMF convergence, one more round is done to see if there are any new beneficial crosslink opportunities for the constellation. Note that both "Xlnk Pairing" and "Exact Time Matching" lines are fractions whose values are shown with the blue axis on the right. "Xlnk Pairing" is the fraction of crosslinks that have met their mutual dependence constraint out of all crosslinks bid and "Exact Time Matching" is the fraction of crosslinks with an exact time match out of the crosslinks that have met their mutual dependence constraint. The dropoffs in the plot occur when crosslinks permission to bid solo parameters reach zero because they have not matched with the mutually dependent partners.

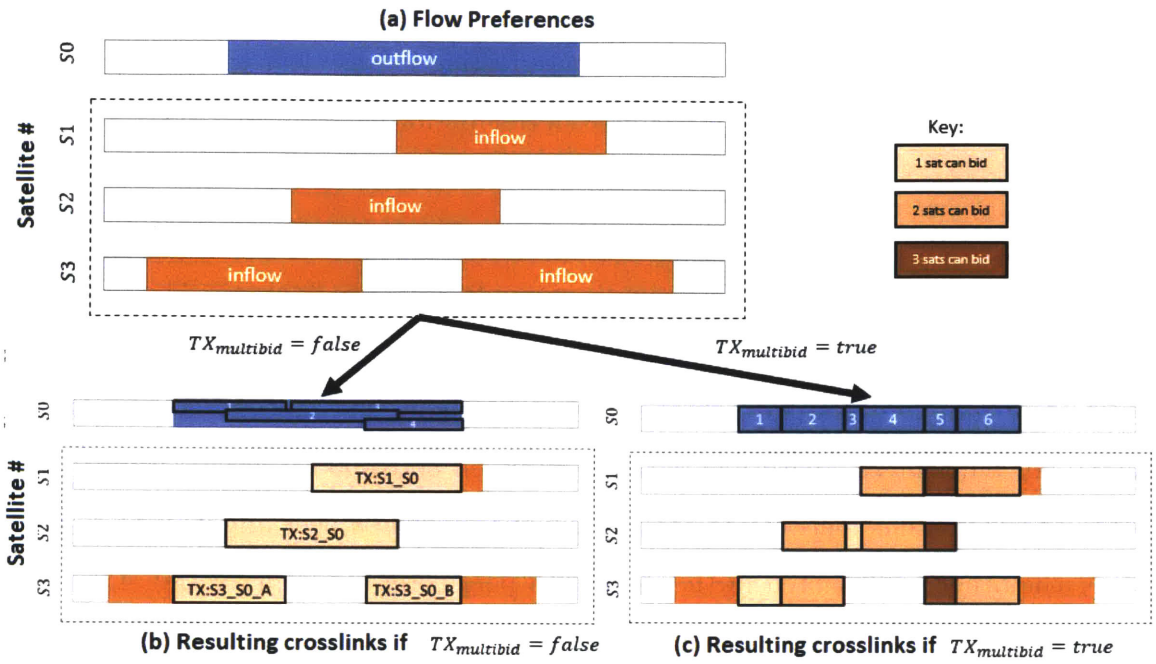


Figure 3-24: Example of TX crosslinks that result from different $TX_{multibid}$ values Note that there are 4 unique TX crosslinks created in the figure on the bottom left, when $TX_{multibid} = false$ and 6 unique TX crosslinks created in the figure on the bottom right, when $TX_{multibid} = true$.

is shown visually in Figures 3-24 and 3-25

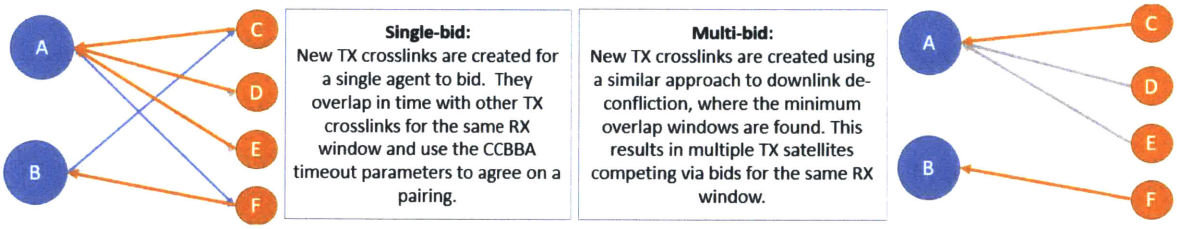


Figure 3-25: Graphical view of single bid vs. multibid TX crosslinks

This view is intended to be a single timestep snapshot. The ' case has multiple RX crosslinks that may not find a match, resulting in lost total DV output, while the multibid case has only one RX crosslink per satellite per timestep, but there may be more total crosslinks to consider which can slow down the consensus phase.

Initially, the hypothesis was that creating multibid transmission crosslinks would always be better for performance and consensus rounds because the RX crosslinks

would not time out unless there was no possible satellite that could add a TX crosslink to its bundle at that time. However, after more testing it was clear that, in general, setting $TX_{multibid} = true$ results in slightly better performance at the cost of more runtime and consensus rounds. The runtime and consensus gap increases because there are more total crosslinks created in the *true* case, especially as more satellites are added. This is shown visually in a simple example in Figure 3-24 (c). The runtime gap also increases as the number of satellites is increased because the powerset-based calculation of unique overlap windows starts to dominate the runtime and becomes infeasible for efficient on-board computation between 60 to 90 satellites. Additional details on the difference between the two settings and when a user might want to use one over the other is presented in the algorithm parameter study in Section 4.2.

3.3.3 ICMF interfaces for on-board planning

This section presents a notional interface for the ICMF algorithm to some of the subsystems onboard a typical small satellite. Figure 3-26 depicts the high level functional interfaces and differentiates between interfaces required during algorithm execution (in blue) and those required before or after planning and scheduling is completed (in yellow). This is intended to be used as a basis for future work on development of an on-board version of the algorithm or for additional development with lower level schedulers that can improve failure recovery performance. ICMF could be used as the planner in a larger autonomy framework, such as the All-Domain Execution and Planning Technology (ADEPT) framework [49] developed by The Charles Stark Draper Laboratory, Inc as a modular framework for autonomous systems.

The functionality of each of the subsystems or functions that would interface to ICMF can be summarized as follows:

- Guidance, Navigation, and Control (GNC) Subsystem: this subsystem is responsible for orbit determination, maintaining knowledge of constellation neighbors, and, if equipped with propulsion, commands for orbit maintenance. ICMF sends the target set and planning horizon to this subsystem and receives the activity windows for all activities and eclipse windows.

- Communications (Comm) Subsystem: this subsystem includes the low-rate radio used for planning communications. The laser communications done with crosslink activities is controlled by the Executor (or Task Manager), which interfaces directly with the payloads.

- Mission Scheduler: this subsystem could be a software agent that is a lower level scheduler used to maintain the schedule produced by ICMF and trigger replanning if the schedule becomes unexecutable. This subsystem also has the responsibilty to interface to the attitude determination and control (ADCS) subsystem at appropriate times to ensure that pointing requirements for upcoming activities are met.

- Executor: this function represents the physical interface to all of the activities in the schedule, including the laser communications payload for crosslinks, radio for downlinks, and observation payload for observations.

- Task Interference Agent: this is an advanced autonomy function that could be used to predict new targets from existing target sets. Some level of on board data processing and/or ground user feedback would be required to implement this functionality.

- Distributed Ground Users: this represents the user set that ICMF would collect target and downlink information from and incorporate it into future planning horizons.

3.4 Greedy Routing Algorithm

The activity schedule produced by ICMF does not provide a unique way to execute the plan. This is because each satellite needs to decide what data to pass over the crosslinks and downlinks in the schedule. This is where data routing is required. This section covers the routing algorithm used to route the data and generate the results in Chapter 4 for median observation latency and median average age of information. First, a brief explanation of why a routing algorithm is required to calculate these

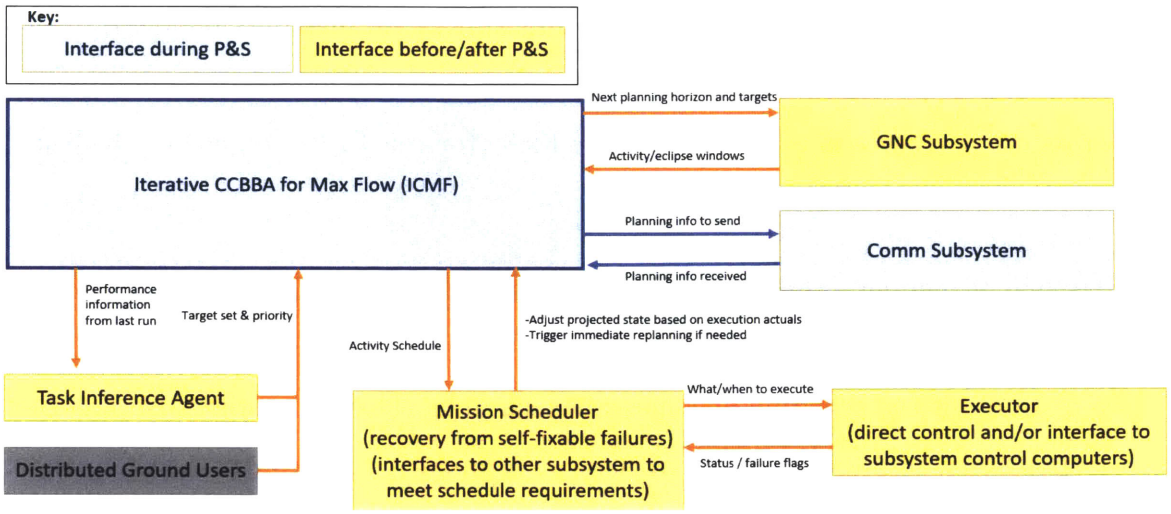


Figure 3-26: Possible ICMF functional interfaces to other satellite subsystems.

metrics is given. Then, the algorithm itself is presented. One of the key assumptions of this routing algorithm is that it is executed after the tasks have all been agreed to and the schedule has been set with ICMF. As discussed in Chapter 2, this is different from many other planning with routing algorithms that either select candidate routes first, or weave them into the planning process [30].

3.4.1 Routing Requirement

Individual observation data packet routing is not taken into account when ICMF creates an activity schedule. Data routing is needed to determine which observation data will be passed across non-observation activities (downlinks and crosslinks) in the schedule. The results from data routing are used to determine statistics relevant to each observation window and unique target. While total data throughput metrics can be calculated without data routing, determining latency for each observation window and the age of information for each unique target requires knowing which observations' data packets are present in each of the downlink tasks, which requires a data routing algorithm. The data latency and AoI metrics are used for two main reasons: 1) they represent metrics that may be important to some users that might prefer timeliness of information as well as total amount of information, and 2) these metrics are used in the SPRINT CGP [30] and can provide additional comparison

points for ICMF against a centralized algorithm that is more route-focused in its approach. The definitions of these metrics are:

- Initial Observation Latency (or latency, lat_{ob}): This metric is a property of each individual observation window (which are converted to observation $Tasks$ at the start of the algorithm) and is simply the time from the end of the observation to the time of the downlink for a specified amount of data from that observation window. lat_{ob} is calculated for each observation window as the difference between the time the first 100 Mb of data was downlinked for that observation and the end time of the observation $ExecutableTask$ ($lat_{ob} = t_{ob}^{dlnk} - t_{ob}^{end}$). A different satellite could downlink the data than the satellite that observed the data, but for this metric to be calculated, at least 100 Mb from an observation $Task$ has to be transmitted in the same downlink $Task$.

- Target Age of Information (or AoI, $G_{targ}(t)$): This metric is a property of each unique target and measures the age of the knowledge on the ground about each target. Calculated for each target and for each timestep in the planning horizon in an iterative fashion for the set of all completed observations of that target. This done in the following steps:

  1. Initialize as linear function from 0 to dur_{hor} (duration of the planning horizon): $G_{targ}(t) = t - t_{hor}^{start}$, where t_{hor}^{start} is the start of the planning horizon and t is the absolute time in the schedule.

  2. For each observation in the set of all observations for that target, compute the following:

  $$G_{targ}(t) = min(t - t_{ob}^{end}, G_{targ}(t)) \quad \forall t \geq t_{ob}^{end} + lat_{ob} \tag{3.7}$$

  3. The final value of $G_{targ}(t)$ after all observations have been accounted for is the AoI for that target. In the loop above (2), a new increasing linear function replaces the previous one and the new linear function starts at 0 and goes up to $dur_{hor} - t_{ob}^{end}$, starting at the ending time of the observation; however, this function only applies after the data has been downlinked,

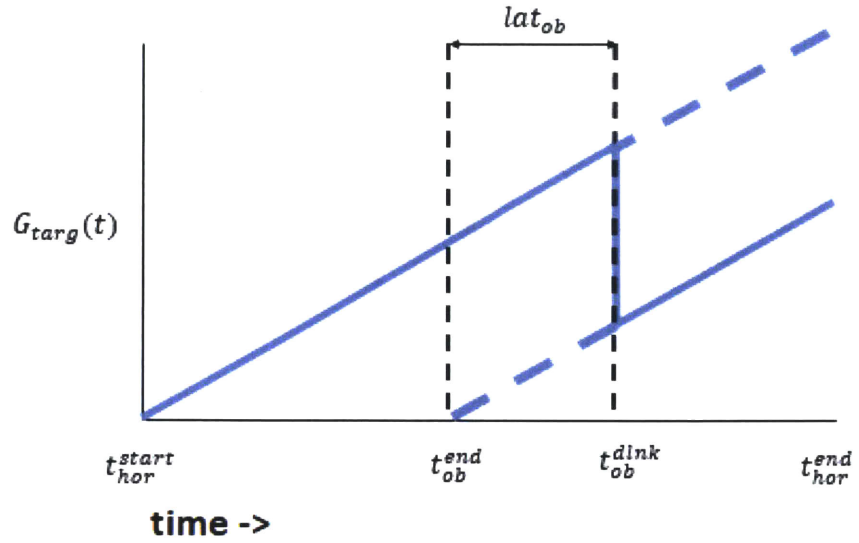


Figure 3-27: Simple AoI example calculation with one observation.
The thick blue line represents the final AoI as calculated with this one observation and the dotted blue lines represent the remainder of the two linear functions that are removed in the composition to compute the final AoI. Except for discontinuous jumps when a downlink occurs, the slope of the AoI function is always 1 (if schedule and AoI time units are the same, which is true in this case).

which is why it is only applicable for $t \geq t_{ob}^{end} + lat_{ob}$ since the time of downlink is equal to the observation end time plus the calculated latency, $t_{ob}^{dlnk} = t_{ob}^{end} + lat_{ob}$. A simple example with only one observation is shown in Figure 3-27.

3.4.2 Minimizing Latency and Age of Information from ICMF Tasks

With the key metrics, observation latency and target age of information, and with the defined data threshold of 100 Mb, a simple greedy routing algorithm was developed to route the data after ICMF determines the activity schedule for all *Tasks*. The solution ICMF provides to the BDRP limits the route options to be those that pass through the linkages provided by the scheduled activities of the constellation.

However, there is still some flexibility for data routing options because of the decision to route the first 100 Mb through a crosslink or a downlink. When the

downlink comes before the crosslink, the decision is easy and the downlink is always chosen. When the crosslink is before the downlink, there is a possibility that the crosslink could result is better latency and AoI performance. Instead of optimally solving this problem with another round of consensus, a routing algorithm that uses a fixed heuristic value is used to address this decision. This heuristic, h_{dlnk}^{bias}, represents the bias to route the 100 Mb of observation data on a downlink available to the satellite instead of sending it on a transmission crosslink. This way, additional communication is not required and each satellite can make their own routing decisions. The value of h_{dlnk}^{bias} was set to 30 minutes for all results in Chapter 4, but it could be any positive number. Each satellite uses its own schedule to see when the next transmission crosslink and downlink are, if the crosslink occurs h_{dlnk}^{bias} minutes or more before the downlink, then that first 100 Mb associated with an observation is routed on the crosslink instead of the downlink. The entirety of the algorithm is shown in the pseudocode block of algorithm 5.

One additional note is that if a first 100 Mb data block comes across in a reception crosslink, then that data block gets priority on the next downlink or crosslink (after the satellite's own "first 100 Mb" blocks). Each satellite does not know until execution time what the routes are, but in all cases tested, no downlink was full from just the first 100 Mb blocks, so as additional priority blocks came across reception crosslinks, they were able to add them to the next available downlink and defer some bulk data to later downlinks. After each of the first 100 Mb blocks are downlinked, then any remaining data is bulk data and not counted for in the latency and AoI statistics, so the routing is simple since there is no preference. The remaining downlink time is filled with any remaining data on-board the satellite executing each downlink.

The inputs to the routing algorithm are as follows:

- $\mathcal{O}$: The set of all executed observations. This is equivalent to the set of all $ExecuteableTasks$ that are on the final bundles of all satellites that have type obs.

- $\mathcal{I}_{PS}$: The set of all satellite agents after ICMF has been run (post-scheduling).

The bundle of these agents collectively set the possible routes that can be executed because they constrain the routing problem to the tasks on the bundle.

- DV_{lat}: The amount of data volume that needs to be delivered for initial latency to be counted for that observation and for target age of information to be updated. This is fixed at 100 Mb for all results in Chapter 4 due to the rationale from Kennedy 2018 that the first 100 Mb provides science users the most critical information [30], but it could be changed based on user needs.

- h_{dlnk}^{bias}: The heuristic bias to downlink instead of crosslink. A crosslink has to occur h_{dlnk}^{bias} minutes sooner (30 minutes in all reasults) than a downlink for it to be chosen for routing.

The first loop of the algorithm, which is a "For loop" over all the observation tasks, initially populates the *openRoutes* and *completedRoutes* sets. These variables contain route ordered set objects, R_o, which always start with an observation. *openRoutes* contains routes that end with a transmission crosslink, which is why they are "open" still. After the first "For loop", these two sets are populated, but the *openRoutes* could still be completed with a downlink by considering the activities on board the receiving satellite from the transmission crosslink. This is the function of the second loop, which is "While loop" because a route could remain in *openRoutes* after several iterations by having more transmission crosslinks added instead of downlinks.

In the second loop, each R_o in *openRoutes* is examined. The receiving satellite agent from the latest transmission crosslink is retrieved and stored as I_{rx}. This satellite's next open outflows are checked with getNextOpenOutflows(I_x, e, DV_{lat}) function. If there are no open outflows, then R_o is removed from *openRoutes* and its data will be routed to ground in the next planning cycle. If there are open outflows still then the same comparison as the first loop is made to determine if the next stage of the route should be a downlink or transmission crosslink. If a downlink is added, then R_o is added to *completedRoutes* and removed from *openRoutes*. Eventually *openRoutes* will be empty and the routing algorithm returns the *completedRoutes*.

The getNextOpenOutflows(I_x, e, DV_{lat}) function is also used in algorithm 5 and requires additional explanation:

- getNextOpenOutflows(I_x, e, DV_{lat}): Inputs are a fully planned agent (I_x), an executed task (e), and the amount of data to route for that observation (DV_{lat}). This function returns a tuple containing two tasks. The first task is the earliest downlink executed by I_x that occurs after e has ended, where the downlink also has at least DV_{lat} of available DV left (not yet routed). The second task is the earliest transmission crosslink (TX) executed by I_x that occur after e has ended, where the TX also has at least DV_{lat} of available DV left. The data already allocated to a particular outflow is tracked with the $outflow.availableDV$ property. Note that this function will return $(null, null)$ if there is no downlink or TX crosslink that meets the criteria specified above. However, for simplicity of the pseudocode, these *null* objects still have the property t_{start}, but it evaluates to 0 in this case.

Note that algorithm 5 doesn't explicitly route data from each observation beyond the first DV_{lat} (100 Mb). This is because this thesis is not comparing final latency for each observation and the remainder of the data can be sent in bulk over the the remaining space in each outflow. Also note in line 23 of algorithm 5 that observations are never routed if the observation has gotten to a point where it is on a satellite that has no available outflows. This is only because of the limited planning horizon and the BDRP has no requirement that all observations are routed within the planning window. These will be downlinked in the next planning cycle. Ways that ICMF can be modified to accomodate a requirement to route all observations within the planning window and further improve latency and target AoI performance is discussed in Chapter 5.

Algorithm 5 Greedy Routing Algorithm Post Activity Selection

1: **procedure** ROUTEDATA($\mathcal{O}, \mathcal{I}_{PS}, DV_{lat}, h_{bias}^{dlnk}$)
2: $openRoutes \leftarrow \emptyset$
3: $completeRoutes \leftarrow \emptyset$
4: **for each** o **in** $\mathcal{O}$ **do**
5: $R_o \leftarrow \{o\}$ ▷ ordered set indicating tasks in a route
6: $I_o \leftarrow$ getObserver(o)
7: $dlnk_{next}, tx_{next} \leftarrow$ getNextOpenOutflows(I_o, o, DV_{lat})
8: **if** $h_{dlnk}^{bias} > dlnk_{next}.t_{start} - tx_{next}.t_{start}$ **then**
9: $tx_{latest} \leftarrow tx_{next}$
10: $R_o \leftarrow R_o \oplus_{end} tx_{latest}$
11: $openRoutes \leftarrow openRoutes \bigcup R_o$
12: $tx_{next}.availableDV \leftarrow tx_{next}.availableDV - DV_{lat}$
13: **else**
14: $R_o \leftarrow R_o \oplus_{end} dlnk_{next}$
15: $completeRoutes \leftarrow completeRoutes \bigcup R_o$
16: $dlnk_{next}.availableDV \leftarrow dlnk_{next}.availableDV - DV_{lat}$
17: **while** $openRoutes \neq \emptyset$ **do**
18: **for each** R_o **in** $openRoutes$ **do**
19: $tx_{latest} \leftarrow$ getLast Element(R_o)
20: $I_{rx} \leftarrow tx_{latest}.RX_{sat}$
21: $dlnk_{next}, tx_{next} \leftarrow$ getNextOpenOutflows($I_{rx}, tx_{latest}, DV_{lat}$)
22: **if** $dlnk_{next} = null$ **and** $tx_{next} = null$ **then**
23: $openRoutes \leftarrow openRoutes \setminus R_o$
24: **else**
25: **if** $h_{dlnk}^{bias} > dlnk_{next}.t_{start} - tx_{next}.t_{start}$ **then**
26: $tx_{latest} \leftarrow tx_{next}$
27: $R_o \leftarrow R_o \oplus_{end} tx_{latest}$
28: $tx_{next}.availableDV \leftarrow tx_{next}.availableDV - DV_{lat}$
29: **else**
30: $R_o \leftarrow R_o \oplus_{end} dlnk_{next}$
31: $completeRoutes \leftarrow completeRoutes \bigcup R_o$
32: $openRoutes \leftarrow openRoutes \setminus R_o$
33: $dlnk_{next}.availableDV \leftarrow dlnk_{next}.availableDV - DV_{lat}$
34: **return** $completeRoutes$

Chapter 4

Results

This chapter covers the results forICMF applied to the bulk data routing problem. Throughout this section comparisons are made to the SPRINT CGP, which is described in detail in Section 2.3.2. This baselines performance of the ICMF algorithm against a state of the art planning tool for crosslink-enabled small satellite constellations. First, data throughput and runtime metrics are presented in Section 4.1 for observation and downlink cases only, which show the feasibility of the approach and highlight the benefits of the pre-coordination phase to help deconflict downlinks. Then, the results of an ICMF parameter study for the test cases are presented in Section 4.2. This parameter study highlights how the algorithm can be tuned by different users depending on their priority for runtime, consensus rounds, and total data throughput. Finally in Section 4.3, the parameters are fixed to the same value for all test cases and the results are presented for all metrics, including median latency and median AoI, which are defined in Section 3.4.

4.1 Observations and Downlinks Only Results

This section covers the performance of ICMF against the SPRINT CGP for six test cases, refer to Table 3.2, when crosslinks are disabled. Without crosslinks, the bulk data routing problem becomes much simpler because the each satellite only needs to coordinate to deconflict downlinks to ground stations. In addition to coordination being simpler, the total number of activities drops significantly, especially for larger

constellations, since crosslinks make up a larger percentage of total possible activities as the constellation size increases (e.g. for the 30 degree inclination case, with 30 satellites there are 1,080 potential crosslink activities out of 1,398 total activities, while with 90 satellites there are 10,248 crosslink activities out of 11,178 total activities). This change simplifies the problem because of the impact on both activity scheduling and routing complexity.

When activity scheduling is framed as a Mixed Integer Linear Program (MILP), no crosslinks means a significantly lower number of decision variables for selecting activities. When scheduling is done with a greedy sequential task allocation algorithm, such as ICMF, then it results in fewer total tasks to loop through every time the next task to add to the bundle is selected. In both the MILP and greedy sequential cases, there are no mutual dependence constraints to enforce between activities, so there is a decrease in constraint size as well. The total number of routes is only $O(N_{dlnks}N_{obs})$, which is a massive reduction from the worst case number of routes when crosslinks are enabled of $O(N_{dlnks}N_{obs}N_u!)$ (Note: $N_u!$ here is number of satellites factorial, which is included in worst case routing complexity because the number of simple paths in graph grows factorially [24]). Testing performance against OD-only cases is useful to validate algorithm performance, even with the reductions to complexity. The reason is that if ICMF performs poorly in OD-only, then it is unlikely to perform well in the crosslink enabled cases.

In all OD-only test cases, both ICMF and the SPRINT CGP were run on the same computer, which has the following specifications:

- Intel(R) Core i5-7300U CPU 2.6 GHz, with 2 cores and 4 logical processors
- 16.0 GB of Random Access Memory (RAM)

4.1.1 30 degree inclination constellation

The first results presented in this section are for the 30 degree inclination case, with either 30, 60, or 90 satellites equally spread across three orbital planes. Recall from Section 3.1.3 that this case is observation dense because all of the observations lie

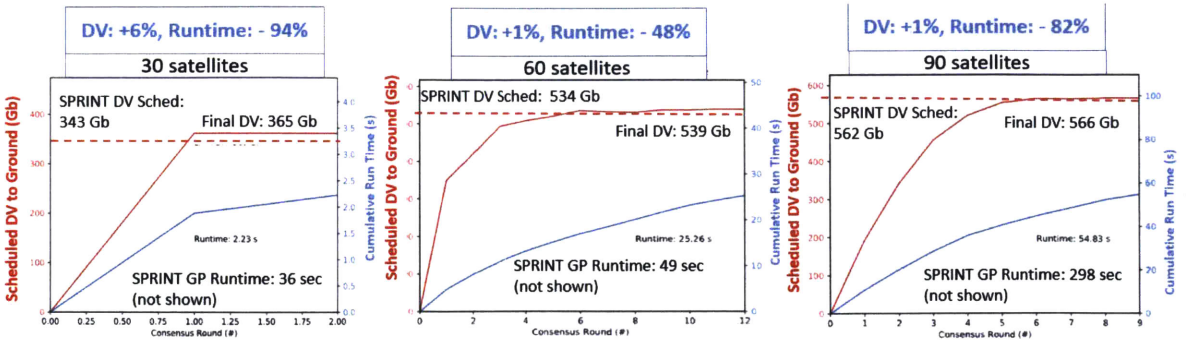


Figure 4-1: Downlink and Observations only results for 30 degree inclination test case.
DV performance and runtime are both favorable compared to the SPRINT GCP, which is summarized in the gray boxes above each subplot. This use case is downlink-limited, so adding additional satellites once ground stations are fully tasked has limited effect on DV throughput

between -30° and +30° latitude, so the constellation is always over the observation locations. This case is also relatively downlink-limited because there are additional NASA NEN ground stations at higher latitudes. This results in the constellation fully allocating much of the downlink time at the 60 satellite size, so there is only a minor increase in total throughput when going to 90 satellites. This also means that ground station deconfliction occurs at almost every possible window so this case is a good test to verify that the pre-coordination phase and downlink deconfliction mechanism for ICMF works effectively. Overall, ICMF resulted in more data throughput and faster runtime than the SPRINT CGP for all three different constellations sizes, as shown in Figure 4-1.

Note that the SPRINT CGP objective function has weights for data throughput, route latency, energy storage, and keeping existing routes. Initially, data thoughput was weighted at 1.0 and the rest were set to 0, but this resulted in runtimes that were over twice as long for the 90 satellite case (compared to what is shown in Figure 4-1), with only a slight improvement to DV. So the objective weights were set to 1.0 for data throughput and 0.1 for the remainder of the factors. This results in near optimal data volume throughput and better runtime performance. The SPRINT CGP activity scheduling MILP also has a 1% optimality gap tolerance so the solutions are within 1% of optimal. SPRINT also uses heuristics to remove routes before activity

scheduling. This may explain how ICMF is able to perform slightly better for data throughput than the SPRINT CGP in this case.

4.1.2 60 degree inclination constellation

This section covers results for observation and downlink only for the 60 degree inclination case. This case is more observation-limited since each satellite only spends a portion of its orbit over the target set. Because of the limited observation opportunities, this test case has less total data volume delivered. As shown in Figure 4-2, ICMF still performs well compared to the SPRINT CGP when it comes to runtime performance with a runtime decrease of -87% on average across the three cases; however, there is a slight drop-off in data throughput performance when compared to the SPRINT CGP. This is probably because this test case has more downlink windows in total, but they are also shorter on average and more unevenly distributed between the different orbital planes for a given start time (see Figures A-2 and A-5 for a comparison between the two constellations' downlink windows). More explicit reasoning about the combinations of observations and downlinks over the entire constellation could result in giving downlink windows to satellites that will hit their data limit later in the schedule.

This type of reasoning is possible in a centralized MILP that considers the combinations of all activities over the constellation. This is is more difficult to implement in a decentralized algorithm that is based on sequential allocation such as ICMF since only high level information is shared in-between task selection rounds. However, data throughput for ICMF is still within 4 % of SPRINT in the worst case, so this is deemed acceptable since there are still large runtime improvements. ICMF is able to stay within a close percentage of SPRINT's data throughput performance because the local value function used for ICMF uses the capacity fraction incentive, f_{DM} (introduced in Section 3.1.2, to capture the majority of instances where a satellite can get more total data volume delivered by outbidding other satellites for a downlink. Additional potential improvements that would allow ICMF to fully close the gap are discussed in Section 5.2.

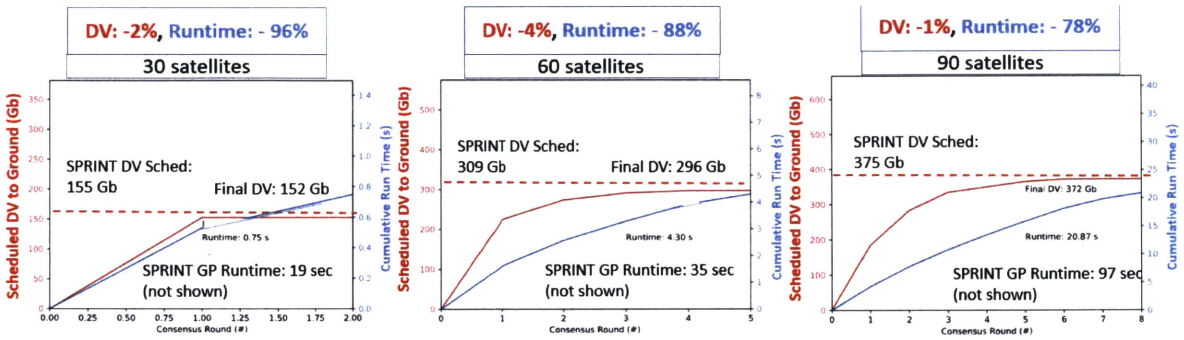


Figure 4-2: Downlink and Observations only results for 60 degree inclination test case.
Runtime is favorable compared to the SPRINT GCP and DV throughput is close, which is summarized in the gray boxes above each subplot. This use case is not downlink-limited to DV throughput continues to increase as constellation size grows.

Before comparing results with crosslink activities, the effect of the ICMF parameters on algorithm performance is explored.

4.2 Selecting Optimal ICMF Parameters for Test Cases

While ICMF was in development, there were several algorithm parameters tested as a way to improve the flexibility of the algorithm. These include a subset of the parameters used for coordinating mutually dependent tasks from CCBBA, such as ω_{ij}^{solo}, which is the permission to bid solo parameter, and, ν_{ij}, which is the timeout parameter for $Task\ j$ stored onboard $Agent\ i$. Algorithm parameters also include the factors described in Section 3.3, such as the number of CCBBA rounds and whether to use multibid transmission crosslinks or not. One could also consider the functions used to create the flow direction preference state as an algorithm parameter since that can be changed modularly and influences different outcomes for the subset of crosslinks considered. To simply the analysis presented in this thesis, an informal study was carried out to fix the CCBBA parameters and the flow direction function for all test cases.

4.2.1 ICMF Algorithm Parameters

Recall from Section 2.3 that the CCBBA parameters need to be tuned for each problem. Generally, there is a tradeoff in runtime and consensus rounds against data

throughput, with number of consensus rounds increasing as ω_{ij}^{solo} and/or ν_{ij} increase. Number of consensus rounds increases because, as ω_{ij}^{solo} and/or ν_{ij} are increased, it allows additional wait time for each task to be in the bundle without its mutually dependent task being selected. However, data throughput doesn't always increase as these two parameters are increased because the coupled task may never be feasible for the other agent(s). Even if data throughput does increase, it is usually a small benefit for a relatively large increase in runtime. After some informal testing to determine which values of the CCBBA parameters resulted in a large amount of DV delivered without unnecessary increases in consensus rounds and runtime, the following values were set for all results presented in this section and remainder of this chapter:

- ω_{ij}^{solo} - Permission to bid solo rounds: set to 1 for all crosslink tasks. Generally, this ranges from 1 to 5, but can be any integer greater than 0.

- ν_{ij}- Time out parameter: set to 3 for reception crosslinks and set to 2 for transmission crosslinks. Generally, this ranges from 2 to 5, but can be any integer greater than 0.

With these values fixed for CMF, which is based off CCBBA with the modifications described in Section 3.2, the parameter study for this thesis focused on the ICMF parameters that govern the iterative loop behavior, which includes crosslink creation type. As mentioned in the Section 3.3, ICMF has two main parameters:

- Maximum Number of CMF iterations (integer): this sets the maximum number of CMF algorithm runs as the inner loop of ICMF. Recall that because crosslinks are created after each CMF convergence, this implies that the number of crosslink creation stages will always be 1 less than this number, so it is also referred to as "Max Iter Xlnks" (for Maximum Iterations for Crosslinks) in some of the plots in Section 4.2.2.

- Create Multibid Tranmission Crosslinks (boolean): if set to true, then the transmission crosslinks created after each CCBBA round are multibid, which requires

using the powerset operation to enumerate all the possible overlap combinations (Many TX per RX). If set to false, then only the crosslink coupled to each reception crosslink that provided value will be created (1 TX per RX), see Figure 3-24.

4.2.2 Parameter Study Setup and Results

The ICMF parameters are varied over a small range and a full factorial comparison is done with all use cases. The parameter values used are:

- $iters_{MAX}$ - Maximum Number of CMF iterations (integer): 2, 3, 4
- $TX_{multibid}$ - Create Multibid Tranmission Crosslinks (boolean): True, False

Each unique use case and parameter setting was repeated 10 times with different random seeds to see how the random ordering of the communication pattern during the consensus phase impacted overall performance. This results in 360 total runs to gather the data for all use cases. The variance from the 10 trials per setting is captured in the error bars present in Figure 4-3.

Figure 4-3 shows the values of data throughput and runtime. Each subplot corresponds to a test case, which can be understood from the title of the subplot. For example, walker30_inc30_NEN corresponds to the 30 degree inclination constellation with 30 satellites. In general, walkerXX_incYY_NEN refers to use case with YY degree inclination and XX satellites. Note that the legend is only included on the top left subplot to avoid cluttering the figure. The key in the legend can be interpreted as follows:

- DV-MB-True: Data throughput values for Multibid = True. (red solid line)
- DV-MB-False: Data throughput values for Multibid = False. (red dashed line)
- RT-MB-True: Runtime values for Multibid = True. (blue solid line)
- RT-MB-False: Runtime values for Multibid = False. (blue dashed line)

In general, setting multi-bid to true results in slightly better DV throughput performance at the cost of a potentially large increase in runtime, as shown in Figure 4-3 by comparing the solid to dashed lines.. The runtime increase is especially true for larger constellation, where the powerset operation is computed for up to 20 potentially overlapping crosslinks and since that operation scales factorially in input size and must be done for each satellite for all overlapping crosslinks, this operation after each CCBBA round converges can start to dominate the runtime. Another general trend is that DV performance increases with number of CCBBA rounds (which is the X-axis in Figure 4-3 as Max Iter Xlnks); however, this trend is not as consistent and sometimes adding a fourth iteration can decrease total DV throughput performance. This is a worrying trend that warrants further investigation as described in Chapter 5. Overall, the parameters provide users with options to trade off additional runtime for a slight increase in total data throughput. The selection of parameters for the final results is explained in Section 4.3.

4.2.3 Bounds on consensus rounds required for convergence

Before moving on to the final results, one additional aspect that is especially import for deploying this algorithm on flight software is the total number of consensus rounds required. Recall from Section 3.3.1 that a consensus round is inclusive of one bundle phase and one consensus phase of CMF. The consensus phase of CMF involves each satellite exchanging consensus dictionaries with all of their neighbors in the communication network until consensus is achieved or round information has propagated through the network (this is controlled by the R_c parameter, see Section 3.2). Knowing a bound of the number of consensus rounds allows users to estimate the energy impact of planning when done on-orbit. Once the amount of data sent per transmission (based on number of tasks) and transmissions per round (based on constellation geometry) are specified, then the energy per round can be estimated. Using this with a bound on the number of rounds allows each satellite to estimate the energy state impact of planning at the start so that there is a lower likelihood of unexpected energy shortage. The results for consensus rounds from the parameter study are presented in

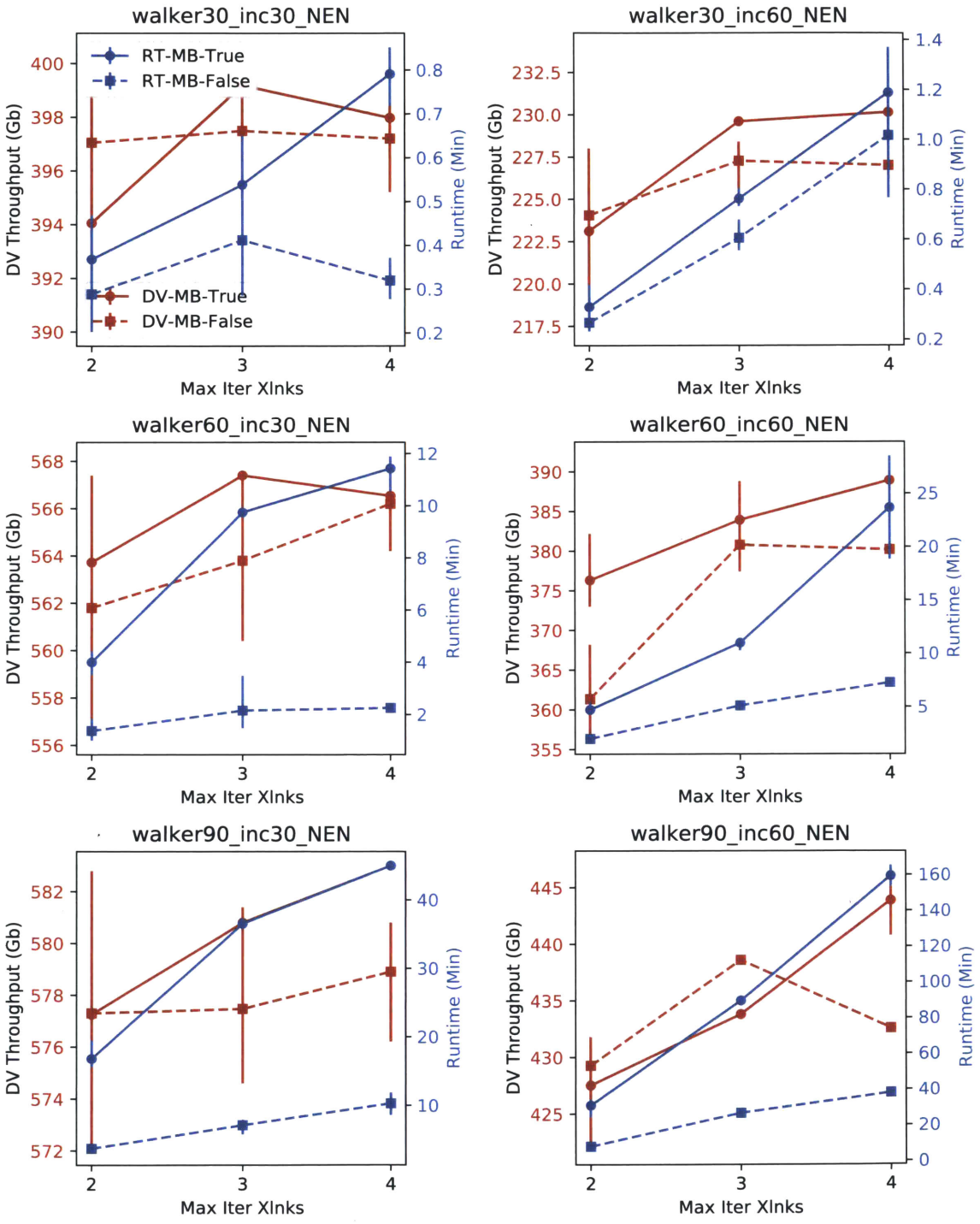


Figure 4-3: Parameter Study: DV and Runtime Performance

Figure 4-4. Additional analyses required for implementing the algorithm for on-orbit planning are discussed in Section 5.2.

Figure 4-4 shows that consensus rounds grows linearly at worst with the number of satellites for all parameter settings tested over all use cases. The variance in consensus rounds is also relatively low, with the maximum and minimum over 10 trials always within 10 rounds of the mean. While this is not a comprehensive analysis, it does provide confidence that implementation on-orbit is possible since the satellites can establish a conservative, but relatively tight, bound on the impact of planning on energy state by taking the maximum value for the desired setting. One final note is that the total number of messages sent grows super-linearly because in each consensus round there are more neighbors to talk with during each round. This raises the number of messages per round linearly with the number of satellites (e.g. for the pre-coordination phase in the 30 degree inclination case, the average messages per satellite are 14.4, 31.8, and 53.2, for 30, 60 and 90 satellites, respectively). The two linear impacts are multiplicative and causes the number of messages scaling as $O(N_u^2)$. Therefore, additional modifications may be required to make the communication and energy impact manageable for large constellations. This important issue is discussed further in Section 5.2.

4.3 ICMF Performance against Centralized Planner

This section covers the performance of ICMF with the greedy routing algorithm of Section 3.4 against the SPRINT CGP over all metrics. The metrics covered include:

- Runtime: Measured in minutes, this is the total time required to go from a set of physical windows ($Tasks$) to a full feasible schedule, which is consistent for the runtime measurement used in the SPRINT CGP. Additional details on the computation platforms used for the comparisons are explained in Section 4.3.1.

- Data Throughput (DV): Measured in Gigabytes, this is the total amount of observation data downlinked to any ground station.

- Median Latency (Med. Lat): Measured in minutes, this metric is calculated by

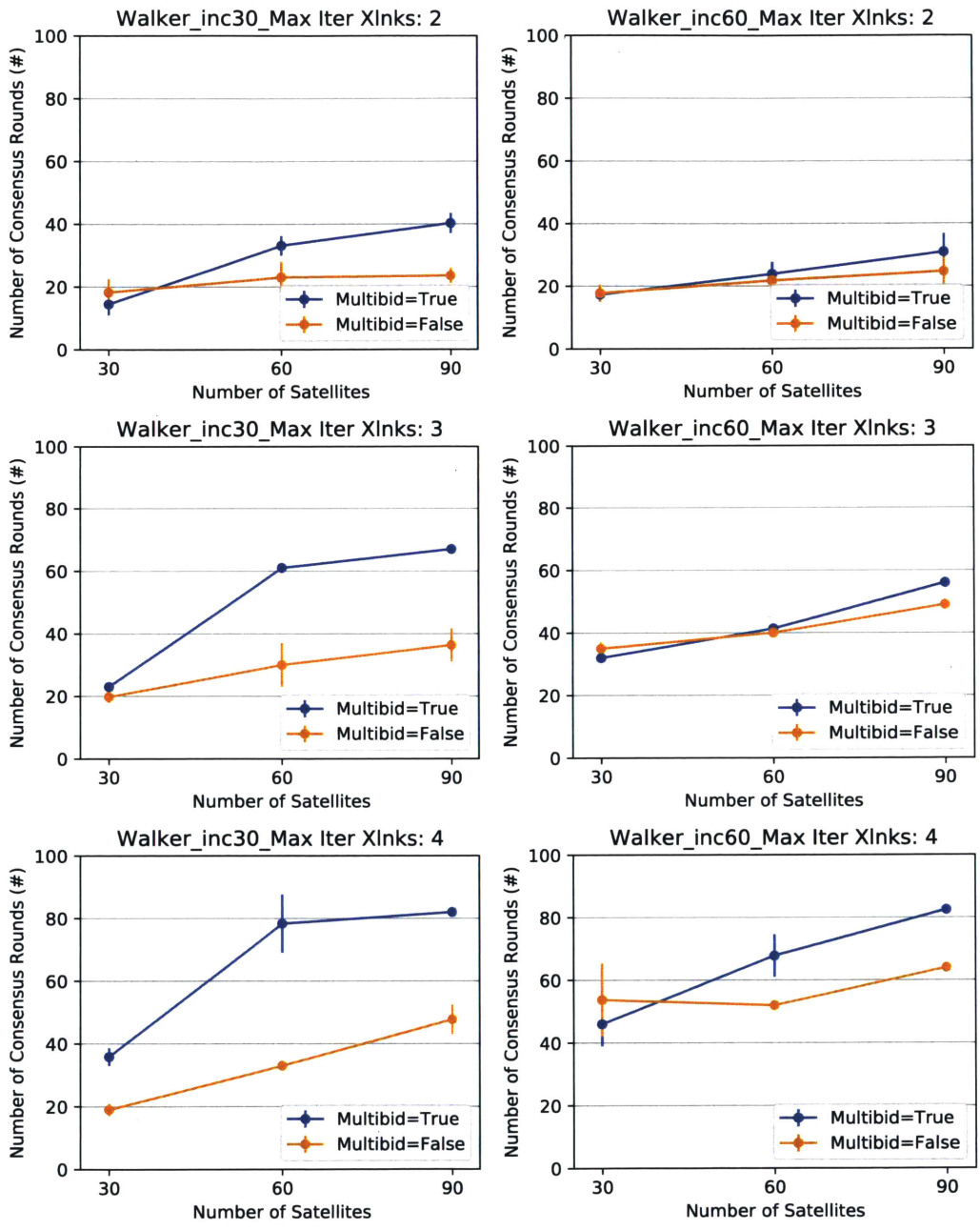


Figure 4-4: Consensus rounds for each unique parameter setting. Parameters tested are shown at the start of Section 4.2.2 and printed in the title and legend of each subplot. This shows linear growth as a function of number of satellites for all test cases and all algorithm parameters, although number of messages grows superlinearly because in each consensus round there are more neighbors to talk with as the constellation density increases

taking the median over all observations latency values. Observation latency is calculated for each observation window as the time from observation to the time of downlink for the first 100 Mb of data.

- Median Average Target Age of Information (Med. Ave. AoI): Measured in minutes, this metric is calculated by first taking the average with respect to time of the age of information for each target. This yields one average value per target. The median of those average values is taken to yield one metric per run. AoI is a property of each target and measures how old the information is on the ground about each target, see Figure 3-27 for an example.

As mentioned in Section 4.2, two parameters of ICMF need to be set to execute the algorithm. As shown in Figure 4-3, the following parameters are chosen to compare performance with SPRINT:

- Maximum Number of CCBBA iterations (integer): 3. This was chosen because 3 always increases Data Volume (DV) compared with 2 iterations for a relatively minor runtime increase. Generally, additional value can be gained by using 4 iterations, but sometimes this also results in worse performance and excessive runtimes.

- Create Multibid Tranmission Crosslinks (boolean): False. False was chosen because of the runtime performance benefit compared with setting it to True.

4.3.1 Centralized Planner Configuration for on the same Test Cases

Before presenting the comparison results, this section presents details on the settings used for the SPRINT CGP. Initially, the SPRINT CGP and ICMF runs were done on the same computer with the following specifications (note: all ICMF runs were done on this computer):

- Intel(R) Core i5-7300U CPU 2.6 GHz, with 2 cores and 4 logical processors

- 16.0 GB of RAM

However, this laptop was unable to run SPRINT for the use cases above 30 satellites, so after the 30 satellite comparisons were completed and ICMF was validated for those use cases, another approach was needed to get performance results for the SPRINT CGP. On another project which focused on improving the scalability of the SPRINT CGP, all test cases were run on an AWS instance that provided enough Virtual Central Processing Unit (vCPU) cores and RAM to be able to solve the problem with a centralized planner. :

Recall that the SPRINT planning paradigm is to first enumerate all the possible routes, then downselect the routes to only consider a subset for planning and scheduling purposes. Because of this approach, the total planning and scheduling runtime includes:

- Route downselection
- Model construction, using Pyomo for the MILP solver interface. (Pyomo is a Python-based open source modeling framework for optimization, see www.pyomo.org).
- MILP solving using Gurobi as the solver. (Gurobi is a commercial optimization software with free academic licenses, see www.gurobi.com).

Runs were done on an AWS m5.24xlarge instance which has a Xeon Platinum 8000 with 3.1 GHz clock speed, 96 vCPU and 384 GiB memory. SPRINT was setup to take advantage of the increased processing power by using parallelization in the following two ways:

- Route down selection cores were 8, 8, 32 for the 30, 60 , and 90 satellites cases, respectively. This part of the SPRINT solution pipeline parallelizes well and receives about a 40% speed up for every doubling of cores.
- Gurobi MILP solver was using 32 cores (96/2, rounded down to nearest power of 2). However, the MILP doesn't benefit nearly as much from parallelization and only receives about a 10% speed up for every quadrupling of cores.

SPRINT CGP Results					
Walker Constellation at 30 degree inclination with 100 targs and 10 NEN ground stations					
Number of sats	Runtime (minutes)	DV (Gb)	Med. Lat (minutes)	Med. Ave. AoI (minutes)	MILP hit 5,000 sec timeout?
30	17.5	403.4	12.74	31.34	No
60	113.5	531.2	9.83	23.67	Yes
90	N/A	N/A	N/A	N/A	Yes, no feasible soln after 10,000 sec
Walker Constellation at 60 degree inclination with 100 targs and 10 NEN ground stations					
Number of sats	Runtime (minutes)	DV (Gb)	Med. Lat (minutes)	Med. Ave. AoI (minutes)	MILP hit 5,000 sec timeout?
30	85.3	262.7	12.87	35.33	No
60	95.78	378.4	10.25	29.72	Yes
90	104.18	330.7	9.24	33.33	Yes

Table 4.1: SPRINT CGP Results for All Use Cases
SPRINT CGP is able to solve the two 30 satellite cases optimally, but hits the 5,000 second optimality timeout in all other cases. Also hit the 10,000 feasibility timeout in the walker90_inc30_NEN case.

The objective function was set to value DV at 1.0, energy at 0.1, latency at 0.1. This was used because using DV at 1.0 actually caused slower runtimes generally with less slightly less DV than putting the other objectives at a lower weighting. The runs also had a MILP "feasibility" timeout of 10,000 seconds, where it would give up if a feasible solution was not found in 10,000 seconds. This happened in the 90 satellite, 30 degree inclination case. The runs had a MILP "optimality" timeout of 5,000 seconds, where it would stop iterating to improve the solution after 5,000 seconds. This happened in 4 of the 6 cases, as annotated in the table below.

The results for the SPRINT CGP for all test cases and all performance metrics are summarized in Table 4.1. Note that the 90 satellite case for the 30 degree inclination constellation was not able to finish within the feasibility timeout window so no results are available in the table. Since the SPRINT results were from a different project which required additional funding to execute and getting results for this use case were not essential, the timeout was not increased and re-run. However, there are enough results to make a suitable comparison to show the feasibility and benefits of using ICMF.

In general, an effort to exactly match the computation resources between SPRINT and ICMF is not done in this thesis. ICMF is intended to be ran on embedded processors in a decentralized way so the computational complexity needs to be limited

to a level much lower than available to that of a centralized planner like the SPRINT CGP. As shown by the difference in processor speed and parallelization, SPRINT had more computational resources for the comparisons that follow in the next two subsections. If computational resources were matched, it is likely that the performance gap would increase significantly since ICMF also lends itself naturally to parallelization because of the decentralized nature of the algorithm (i.e. bundle phases can be completely parallelized). Another note is that the routing algorithm from Section 3.4 is not included in runtime numbers for ICMF, but this routing algorithm runs in under 30 milliseconds for the 30 satellite cases and scales linearly with number of satellites because it is looping over number of observations twice, so should never take more than 1 second, which was confirmed for the 60 and 90 satellite cases. The next two subsections cover the results by constellation type (30 or 60 degree inclination).

4.3.2 30 degree inclination constellation

For the constellation with orbital planes inclined at 30 degrees, the performance of ICMF as a function number of satellites is shown in Figure 4-5. A total of 10 trials were executed with the algorithm parameters fixed as described in the previous section. Figure 4-5 provides insight into how ICMF performance scales over all metrics for different numbers of satellites. Note that the latency and AoI metrics are rather flat with ICMF, while SPRINT decreases both metrics as constellation size increases. This is because there is no explicit incentive in the problem to address these metrics so the additional routing opportunities present with more satellites is only exploited for bulk data transmission and not latency nor AoI. However, the iterative consensus approach of ICMF lends itself to incremental consolidation of information such that a tentative routing plan could be built after each CCBBA convergence and the information from that routing plan could influence the value function of the new set of crosslinks. See Section 5.2 for more details.

Tables 4.2 and 4.3 present more direct comparisons to SPRINT performance. Table 4.2 highlights the difference between ICMF and SPRINT for each metric. As mentioned before, SPRINT is able to exploit improvements from additional routes

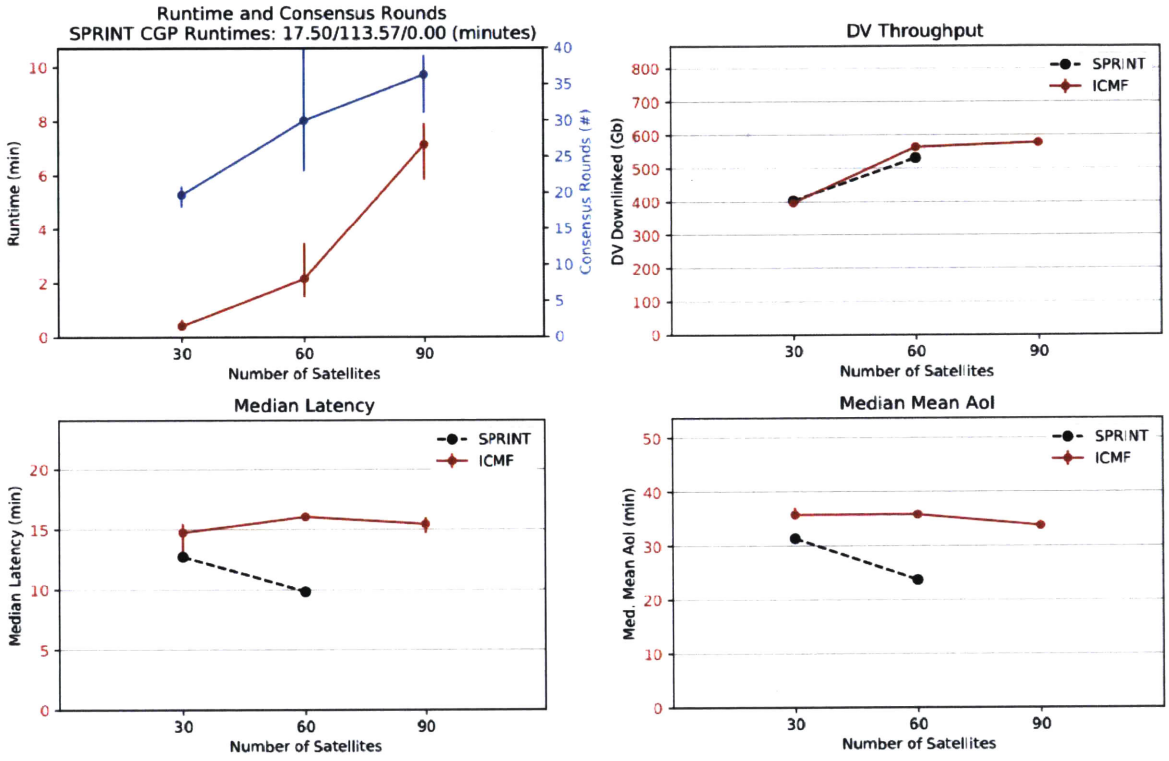


Figure 4-5: Metrics and Runtime Results for 30 degree inclination constellation Note that SPRINT results for runtime are included as a subtitle for the top left plot to preserve axis scale so that the ICMF runtime trend can be shown. All SPRINT results are in Table 4.1

Relative performance: (ICMF - SPRINT)				
Walker Constellation at 30 degree inclination with 100 targs and 10 NEN ground stations				
Number of sats	Runtime (minutes)	DV (Gb)	Med. Lat (minutes)	Med. Ave. AoI (minutes)
30	-17.09	-5.90	2.01	4.38
60	-111.34	32.61	6.23	12.10
90	N/A	N/A	N/A	N/A

Table 4.2: 30 deg inclination results: comparison to SPRINT in same units
Note: the text is colored red if SPRINT is better in that metric and colored blue if ICMF is better. N/A means that ICMF achieved results within timeout, but SPRINT did not, so no comparison is made.

Performance Multiplier: ICMF/SPRINT (if higher is better), or SPRINT/ICMF (if lower is better)				
Walker Constellation at 30 degree inclination with 100 targs and 10 NEN ground stations				
Number of sats	Runtime	DV	Med. Lat	Med. Ave. AoI
30	42.68	0.99	0.86	0.88
60	52.55	1.06	0.61	0.66
90	N/A	N/A	N/A	N/A

Table 4.3: 30 deg inclination results: comparison to SPRINT as a performance multiplier
Note: larger than 1 means ICMF performs better on that metric and lower than 1 means SPRINT performs better, with the magnitude of improvement represented by the multiplicative difference. N/A means that ICMF achieved results, but SPRINT did not within the timeout, so no comparison is made.

to improve latency and AoI metrics as the constellation size grows. However, ICMF is not that far behind in the 30 satellite case, with average latency only 2 minutes longer than SPRINT.

Table 4.3 shows the relative performance difference using a "performance multiplier". This performance multiplier is used because it is easy to interpret for algorithm comparison purposes. Larger than 1 means ICMF performs better on that metric and lower than 1 means SPRINT performs better, with the magnitude of improvement represented by the multiplicative difference. For example, in the 60 satellite case for this constellation, the table shows that ICMF runs 52 times faster, achieves 1.06 times the data volume, but only 0.61 the latency performance and 0.66 the AoI performance.

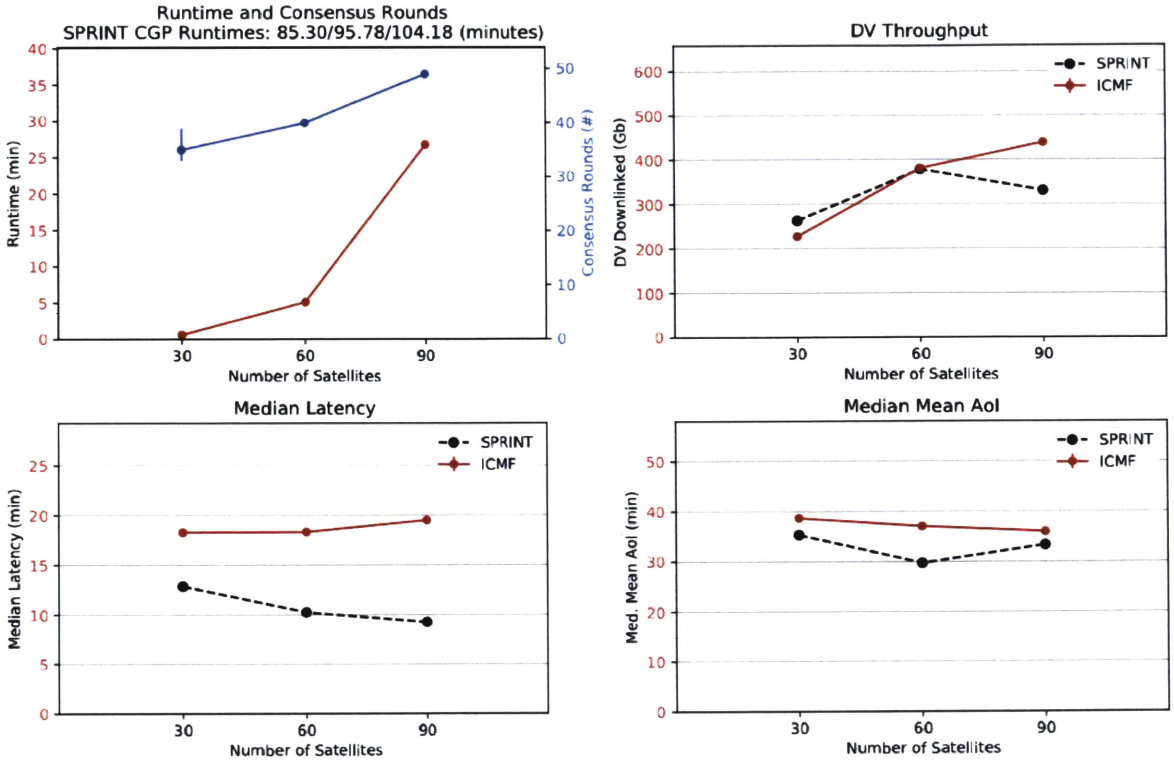


Table 4.4: Metrics and Runtime Results for 60 degree inclination constellation Note that SPRINT results for runtime are included as a subtitle for the top left plot to preserve axis scale so that the ICMF runtime trend can be shown. All SPRINT results are in Table 4.1

4.3.3 60 degree inclination constellation

This subsection presents the same format of results as Section 4.3.2, for the constellation inclined at 60 degrees. The same trends described in the previous subsection hold here as well.

The latency benefit of SPRINT over ICMF increases as the constellation size grows, as seen with the addition of the SPRINT result for the 90 satellite constellation.

The contribution of ICMF remains as a near optimal data throughput planner that can run on average 50 times faster than the SPRINT centralized global planner for small satellite constellations. However, there are several possible areas of improvement that could yield improved latency and AoI performance while still maintaining the runtime advantage. In addition to drawing some conclusions from these results, these improvement methods, as well as some necessary modifications to make flight software

Walker Constellation at 60 degree inclination with 100 targs and 10 NEN ground stations				
Number of sats	Runtime (minutes)	DV (Gb)	Med. Lat (minutes)	Med. Ave. AoI (minutes)
30	-84.69	-35.47	5.41	3.31
60	-90.67	2.42	8.08	7.30
90	-77.44	107.92	10.26	2.62

Table 4.5: 60 deg inclination results: comparison to SPRINT in same units
Note: the text is colored red if SPRINT is better in that metric, and colored blue if ICMF is better.

Performance Multiplier: ICMF/SPRINT (if higher is better), or SPRINT/ICMF (if lower is better)				
Walker Constellation at 60 degree inclination with 100 targs and 10 NEN ground stations				
Number of sats	Runtime	DV	Med. Lat	Med. Ave. AoI
30	139.84	0.86	0.70	0.91
60	18.74	1.01	0.56	0.80
90	3.90	1.33	0.47	0.93

Table 4.6: 60 deg inclination results: comparison to SPRINT as a performance multiplier
Note: larger than 1 means ICMF performs better on that metric and lower than 1 means SPRINT performs better, with the magnitude of improvement represented by the multiplicative difference.

implementation possible, are discussed in the next chapter.

Chapter 5

Conclusion

ICMF addresses the challenge comes exponential complexity growth when considering all the possible data routes, by planning the activities first for near optimal data throughput, which reduces the possible data routes. There has also been growing interest in decentralized planning and scheduling capabilities to facilitate more autonomous constellations. While ICMF in its current form is implemented on a single CPU and simulates synchronous decentralized execution, this chapter discusses how it can be feasibly deployed on-orbit.

5.1 Contributions Summary

This thesis introduced a new decentralized algorithm called ICMF that can efficiently create a schedule to solve BDRPs for satellite constellations. The functionality of ICMF is described in Chapter 3 and comparisons to a state of the art centralized planner (SPRINT CGP) are presented in Chapter 4. Across the six different test cases used in this thesis, ICMF demonstrated the following:

- ICMF runtime is 3.9 to 139.8 times faster than SPRINT CGP runtime.
- ICMF data throughput is 0.86 to 1.33 times that of the SPRING CGP.
- ICMF with the routing algorithm described in Section 3.4 results in median latency values that are 2.01 to 10.26 minutes longer than the SPRINT CGP median latency values.

- ICMF with the routing algorithm described in Section 3.4 results in median average AoI values that are 2.62 to 12.10 minutes longer than the SPRINT CGP median average AoI values.

The runtime comparisons are with no parallelization implemented for ICMF, while the SPRINT CGP is using a significant amount of parallel processing, as described in Section 4.3.1. With the current implementation, if a ground based planning user values the ability to quickly return a solution that is near optimal for data throughput, then ICMF is a good choice. Additionally, because ICMF is a decentralized algorithm, it naturally lends itself to parallelization when used as a ground based planner. Each bundle phase can be completely done in parallel and parts of the each communication round in the consensus phase can be done in parallel, with some inter-process communication at key points in the communication network.

5.2 Future Work

This section outlines the future work items to improve ICMF. These improvements are along two different lines. Section 5.2.1 describes algorithm improvements that will benefit all users of ICMF, whether running it on a ground station network or running it on different satellite agents on-orbit. Section 5.2.2 explains some of the modifications needed to make implementation of ICMF in flight software on-orbit feasible and effective.

5.2.1 ICMF Improvements

In this subsection, additional analysis and improvements are described that could improve ICMF performance. First, research directions that could further increase the data throughput are described. Then, additional work items to make ICMF route aware, in a computationally efficient manner, are explained. Making ICMF aware of the current set of routes will provide opportunities to increase latency and AoI performance.

Data Throughput Improvements

In Figure 4-3 there were some occurences of total data throughput decreasing as the $iters_{MAX}$ parameter increased from 3 to 4. When $iters_{MAX}$ is set to 4, this means that there will be 3 rounds of crosslink creation instead of 2, and CMF will be run 4 times. Generally, this is not a problem because each user can tune the parameters to values that work best on average for their constellation size. However, in the cases tested in this thesis, data throughput decreased in 3 cases out of 12. There are12 cases because each of the 6 test cases is run once for $TX_{multibid} = true$ and once for $TX_{multibid} = false$. The decrease in data throughput as a function of $iters_{MAX}$ should be improved because a user should be confident that as they increase the $iters_{MAX}$ parameter, performance is non-decreasing, although runtime may increase significantly. To investigate this further, ICMF could be run with a flag to log whenever any of the bundles decrease below the set that was finalized at the end of CMF round 3. Nominally, this should not happen, but sometimes disagreement in consensus phase can cause invalid data states at the start of the next bundle phase, so the bundle needs to be rebuilt from empty. This is not usually a large impact because the most valuable tasks will still be selected again, but there could be occurrences where some key crosslinks were near their last bid solo values and this bundle reset could prevent them from matching again in CMF round 4.

Another interesting research direction to improve data throughput would be to evaluate modifying the value function of a downlink based on both the satellite's current data level and the ratio of observations to downlinks available in the rest of the planning time horizon. Recall the issue noted in the observation and downlink only case for the 60 degree inclination constellation in Section 4.1.2. This issue was a performance drop relative to SPRINT because the SPRINT CGP performs more explicit reasoning about the combinations of observations and downlinks over the entire constellation. This could result in giving downlink windows to satellite that will hit their data limit later in the schedule. Since ICMF is a sequential task allocation algorithm, it only reasons about one task a time (except for task outflow

coupling) based on its current bundle and knowledge of other agents. The data throughput performance gap comes in because of the downlink deconfliction where only one satellite can downlink at a time. Normally, the heuristic that whichever satellite is more full should win the downlink is effective. However, if the satellite that loses the bid for the downlink is currently less full at the time of the downlink, but has a higher observation to downlink task ratio in the future, then total throughput might be decreased if the fuller satellite wins the downlink. Thus, research into new value functions that can dynamically account for remaining activities could help improve total constellation throughput.

Route Aware Improvements

The largest improvements to ICMF can come in the form of closing the gap in the latency and AoI metrics. To do this, some level of route awareness needs to be brought into the algorithm. There are two approaches discussed in this subsection. The first leverages the iterative nature of ICMF to create pseudo-routes from the current activity set at the end of each CMF round. The second shares additional information during each consensus phase. These methods can be combined to be even more effective.

After each CMF execution, the satellite constellation has achieved consensus on a feasible plan. This means that each satellite could execute the routing algorithm based on the current plan to gain some awareness. These are not the final real routes, so they will be called pseudo-routes. These pseudo-routes can be used to modify the value functions for new tasks. For example, after the first CMF iteration (where only observations and downlink $Tasks$ were scheduled), satellites with with no downlinks will have no completed routes for their observations. This means any TX crosslink they receive from the satellites creating RX-TX pairs should be more valuable if it occurs after one or more of their observations. This information can be used after each round to place higher value on TX-RX crosslink pairs that provide routes for non-routed observations or for observations that have existing routes, but the latency score is near the maximum value. The overhead of executing this additional

information sharing should be small. There will be the routing step, which can be executed in under 1 second for all observations, and one additional consensus round before entering CMF, to give the opportunity for both the TX and RX crosslinks to adjust their values accordingly. This is an open research direction that could provide interesting heuristics to give users the ability to tune performance toward the metrics they care most about, while still preserving runtime benefits.

There are also modifications that could be made inside each CMF execution. Additional data could be shared as new crosslinks are agreed on that include partial routing information and do not require strong consensus. This could facilitate improved crosslink matching during execution and crosslink creation at the end of the CMF execution. One example of a weak consensus mechanism that helps share information about observations for planning purposes is the Limited Communication Constellation Coordinator (LCCC) [29]. This tool did not plan for laser communication crosslinks, but it showed improvements in observation and downlink task allocation by sharing information about observation target plans without enforcing strong consensus, so no additional runtime requirement was imposed.

5.2.2 On-orbit Planner

The ability to have an autonomous constellation execute planning and scheduling on-orbit is becoming a larger priority for both commercial and military space applications. A distributed set of users constantly tasking the constellation without a centralized pooling location is a challenge for current centralized planners. However, decentralized task allocation algorithms fit naturally for this purpose. CBBA based algorithms can easily accomodate additional tasking, add them to the current bundle, and pool them until a threshold has been reached to execute a new planning horizon. Or, if they are high enough priority, they can trigger a total replan where activities are released and replanned. ICMF inherits these benefits from CBBA and is still effective at accommodating distributed user requests. There is also the challenge of ensuring that all satellites can communicate with neighbors while sharing the same radio spectrum. To ensure feasible execution on-orbit the amount of energy

using during the planning phase needs to be accounted for so that the plan does not immediately become infeasible due to the starting energy state no longer being accurate.

As mentioned in Section 4.2.3, the upper bound on energy required to complete planning, $\hat{E}(\hat{N}_c, R_c, D_p, N_p, \rho)$, can be estimated with the upper bound on number of consensus rounds, $\hat{N}_c$, and where R_c is the number of communication rounds per consensus round, D_p is the data transmitted per communication partner, N_p is the number of communication partners each round, and ρ is the satellite's communication subsystem parameters which allows conversion from total data to transmit to an estimate of energy required. As the constellation side grows, the total number of messages sent grows polynomially because in each consensus round there are more neighbors to talk with (N_p and N_c both grow linearly and are multiplicative together for number of total messages). To decrease $\hat{E}$ for a given satellite constellation using ICMF, the main variables to adjust are R_c, D_p, and N_p.

To adjust these variables, this thesis outlines two main types of modifications in the communications structure and/or the communication data:

- **Communication Structure:** Currently, each satellite communicates with all satellites that are within range of radio communications (N_p is all satellites in radio range). This is likely sharing a lot of redundant information, especially for dense constellations. Some different communication structures within the communication graph should be investigated, such as:
  - In-plane immediate neighbors plus x out of plane neighbors: This structure would have a parameter x that controls the number of x closest out of plane satellites to communicate with during each consensus round. Keeping x large would result in a lot of crossplane communication and minimal in-plane communication; however, information could still pass quickly in plane after a few rounds if there is a dense crossplane network.
  - Nearest x neighbors, plus y furthest neighbors: This structure would have two parameters where the closest x neighbors are included in communica-

tion, while also including the furthest y neighbors. This structure could try to exploit the small world effect that relies on small groups having a few far-flung connections with other groups in the graph to share information efficiently.

  - A combination of one of the two plans above with changing R_c, number of communication rounds, as a function of estimated convergence state. Satellites can estimate how close the algorithm is to convergence by looking at the percentage of matched mutually dependent crosslinks and time-matched crosslinks, as shown in Figure 3-23.

- **Communication Data:** Additional modifications to the amount and type of data shared between satellites should be investigated to limit the data passed to only what is essential. This will take some investigation and analysis to determine what is essential for each problem because any information that is not shared might risk potential conflict at a later point.

With these adjustments, the value of $\hat{E}$ can be lowered to a manageable level for small satellites to execute on-orbit planning. An accurate bound on $\hat{E}$ also implies an upper bound on the planning time, $\hat{t}_p$. Knowing $\hat{t}_p$ allows the constellation to determine when they need to start the next planning and scheduling session. These adjustments, along with the ICMF algorithm improvements described in Section 5.2.1 should provide a strong basis for a flight software ready algorithm to solve BDRPs with small satellite constellations.

Appendix A

Full Schedules

Here is where the full schedules from each of the main results runs are shown (30/60 sats for inc 30 and inc 60 using ICMF). Four schedules total are presented. The schedules for the 90 satellite cases are available upon request and do not fit well on this paper size. For all four schedules, the legend used for the different types of possible and executed activities is shown below in figure A-1. This legend is only included here and not next to each individual schedule.

Figure A-1: Legend for satellite schedules.

The "Poss" tag indicates a physical window that could be scheduled. If it was scheduled, then the hatched "Exec" tag will be overlaid. This is only applicable for Observations and Downlinks, because there is always possible crosslink windows to at least one satellite.

A.1 30 degree inclination constellation

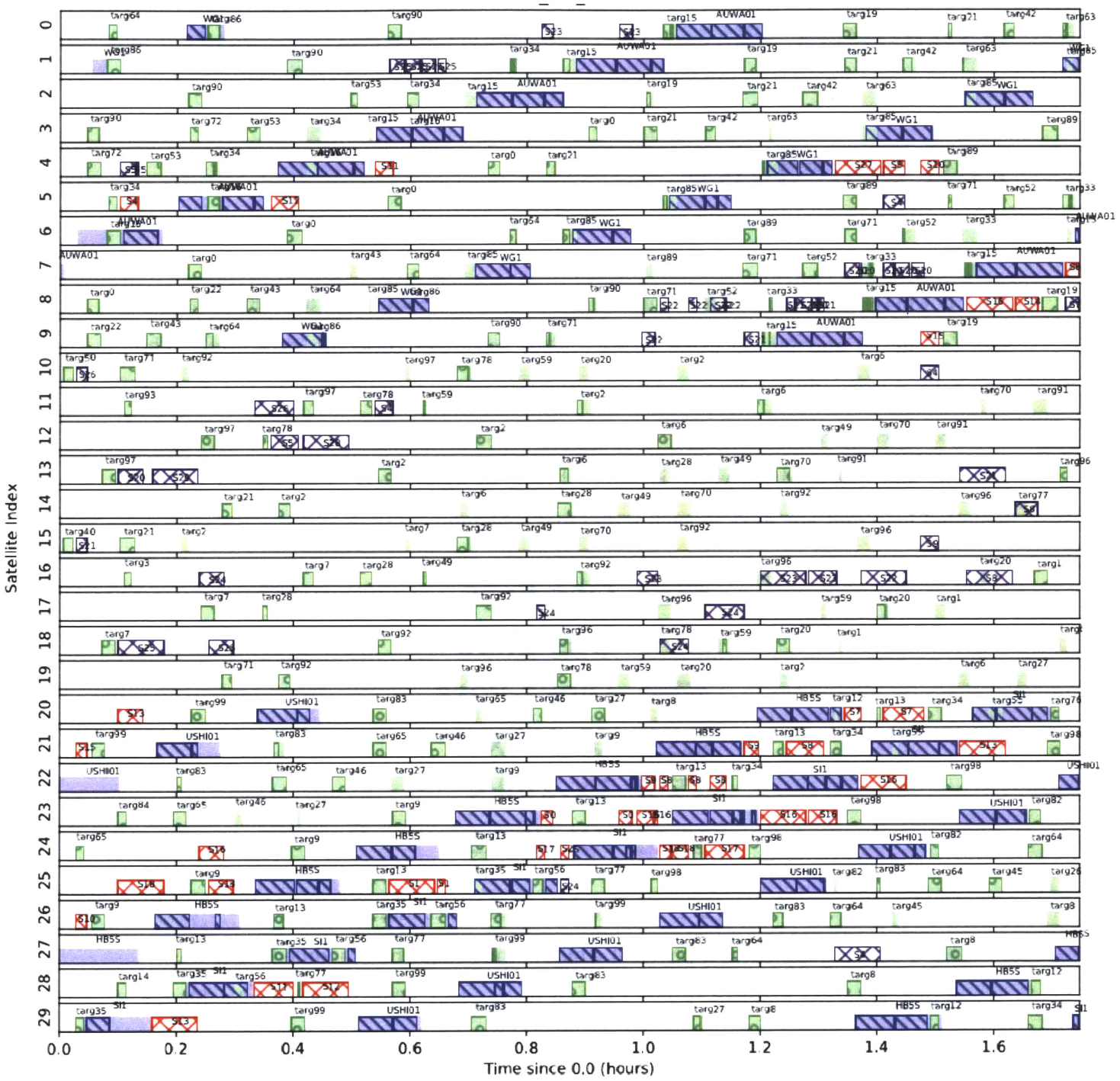


Figure A-2: 30 satellites: Full Schedule for 30 deg inclination constellation.

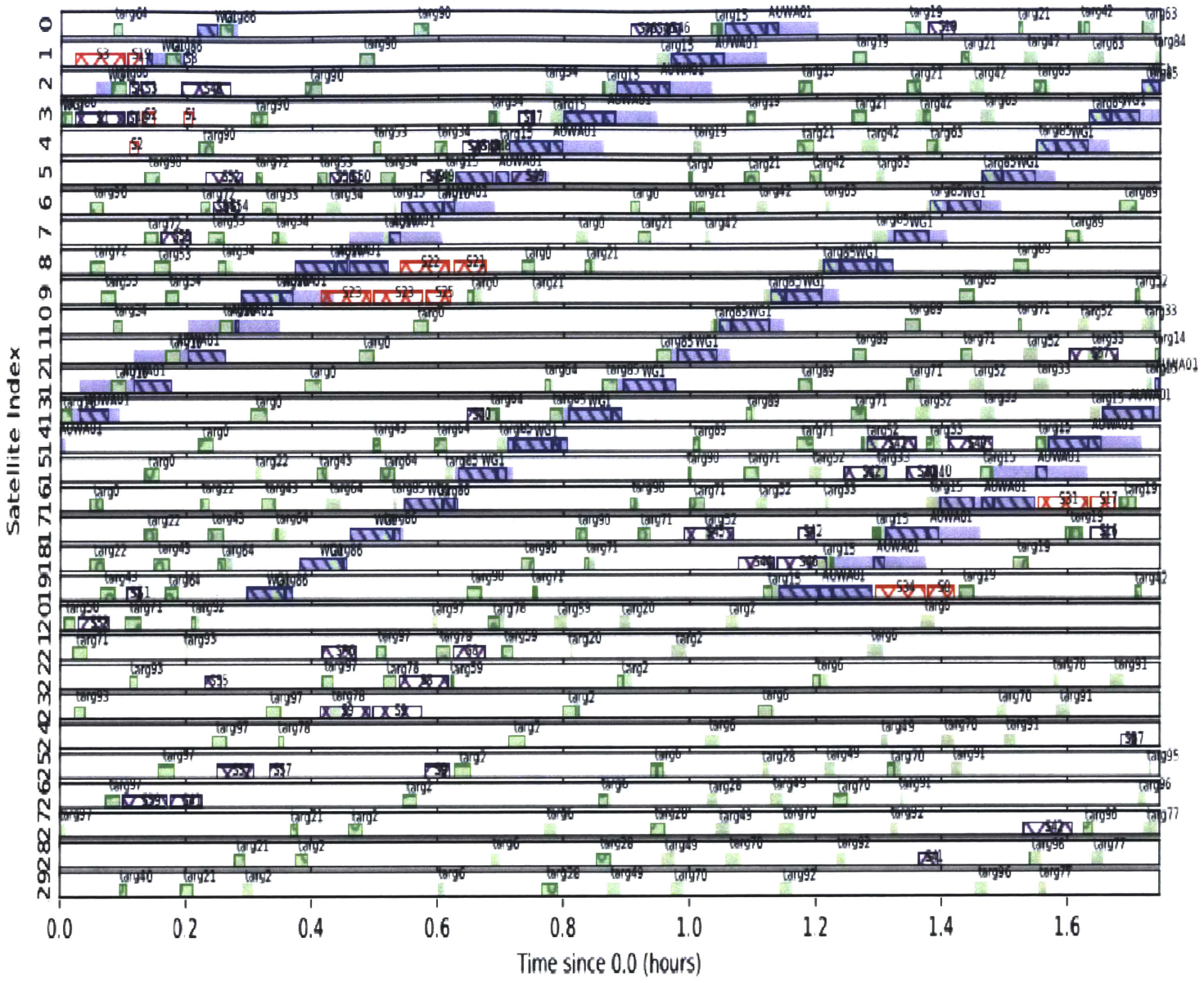


Figure A-3: 60 satellites: Full Schedule for 30 deg inclination constellation. (satellites 1-30)

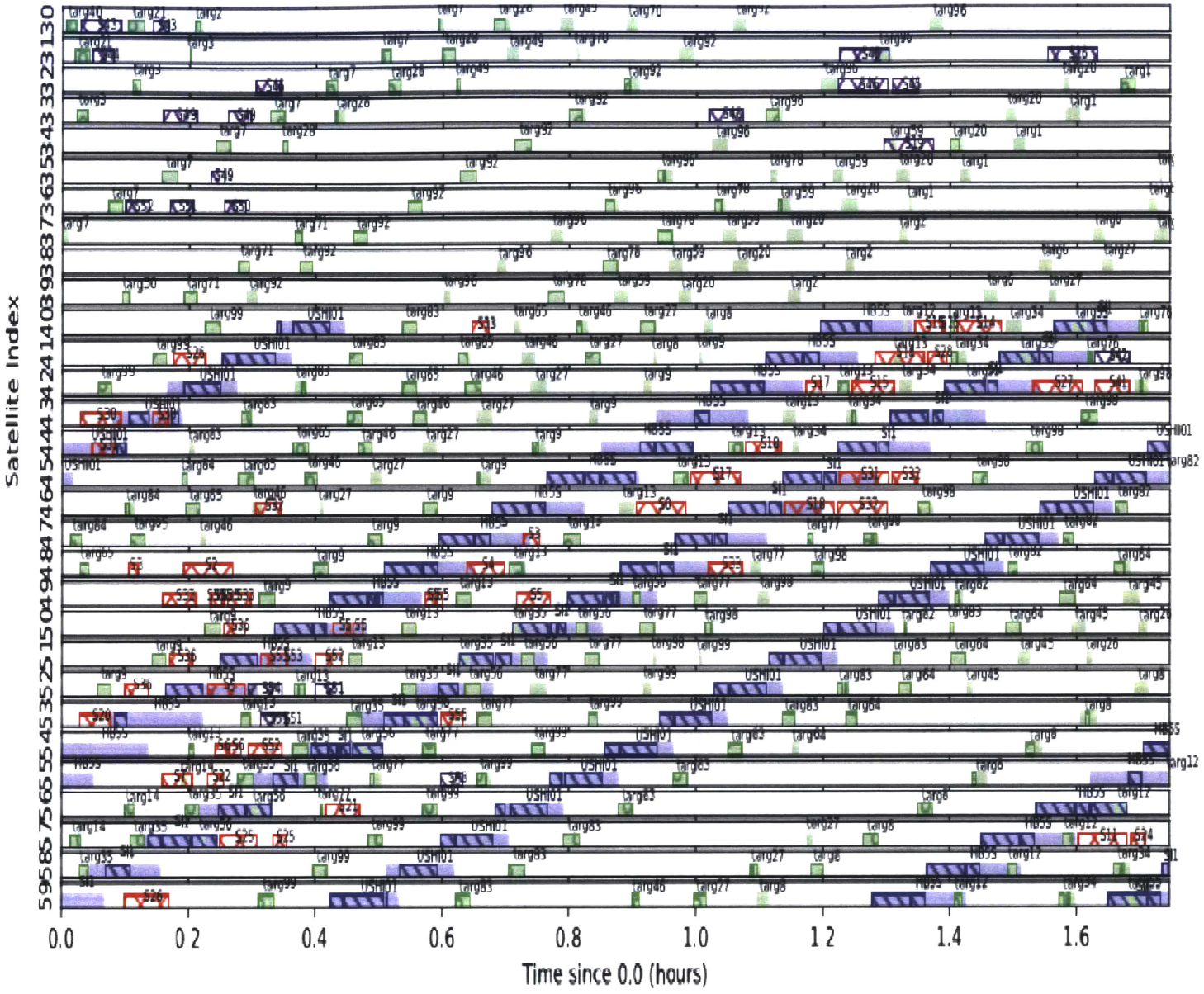


Figure A-4: 60 satellites: Full Schedule for 30 deg inclination constellation. (satellites 31-60)

A.2 60 degree inclination constellation

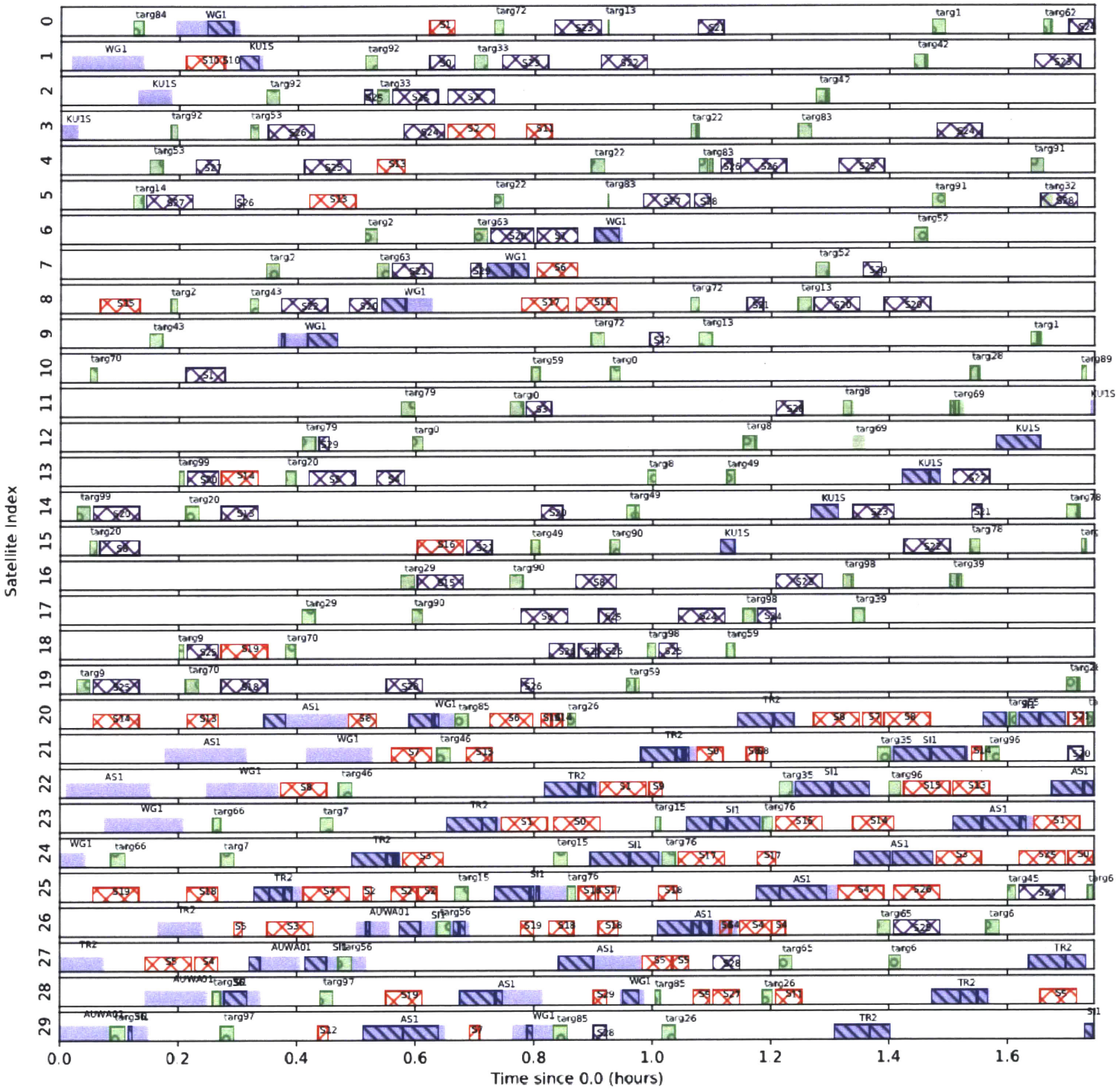


Figure A-5: 30 satellites: Full Schedule for 60 deg inclination constellation.

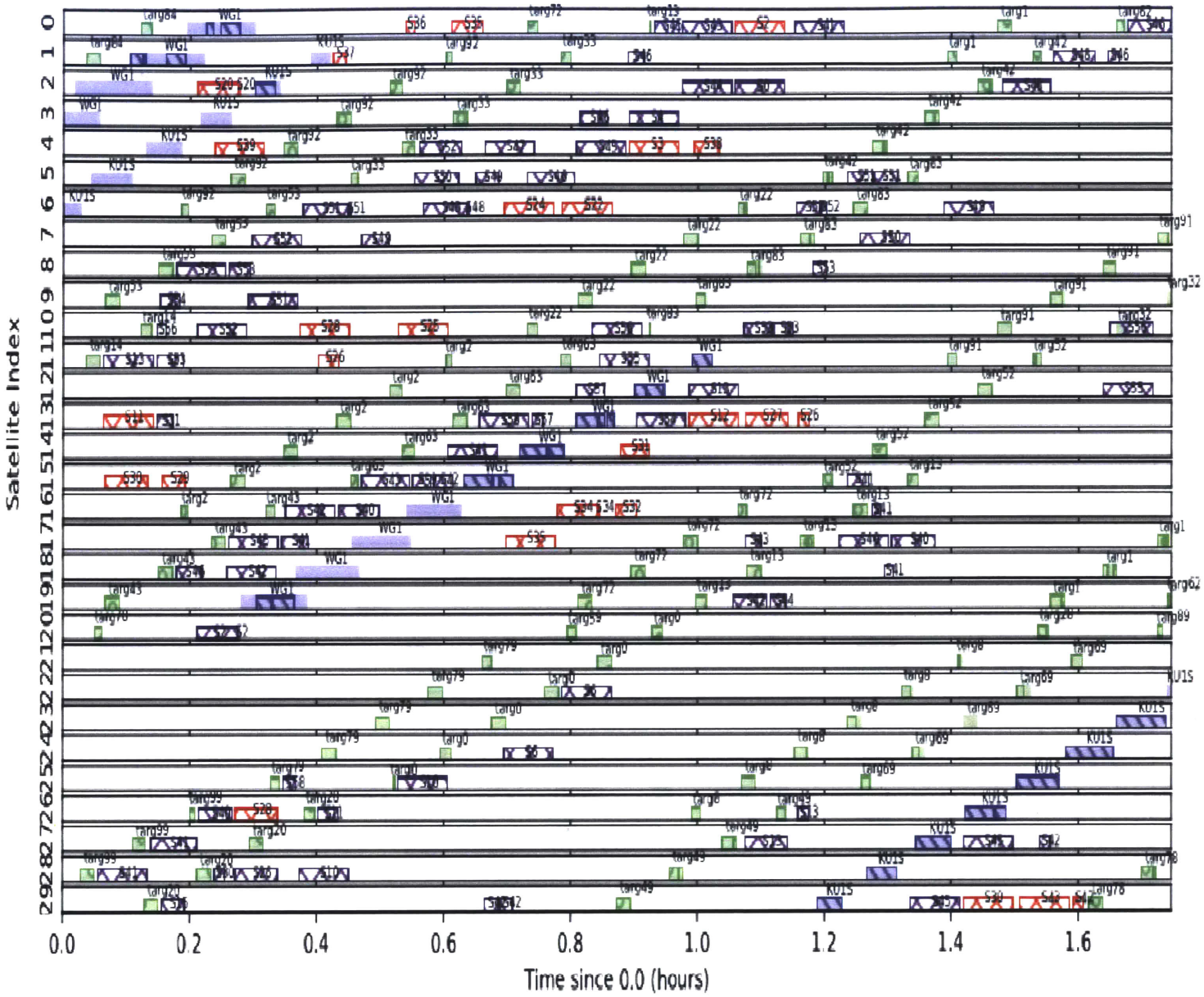


Figure A-6: 60 satellite: Full Schedule for 60 deg inclination constellation. (satellites 1-30)

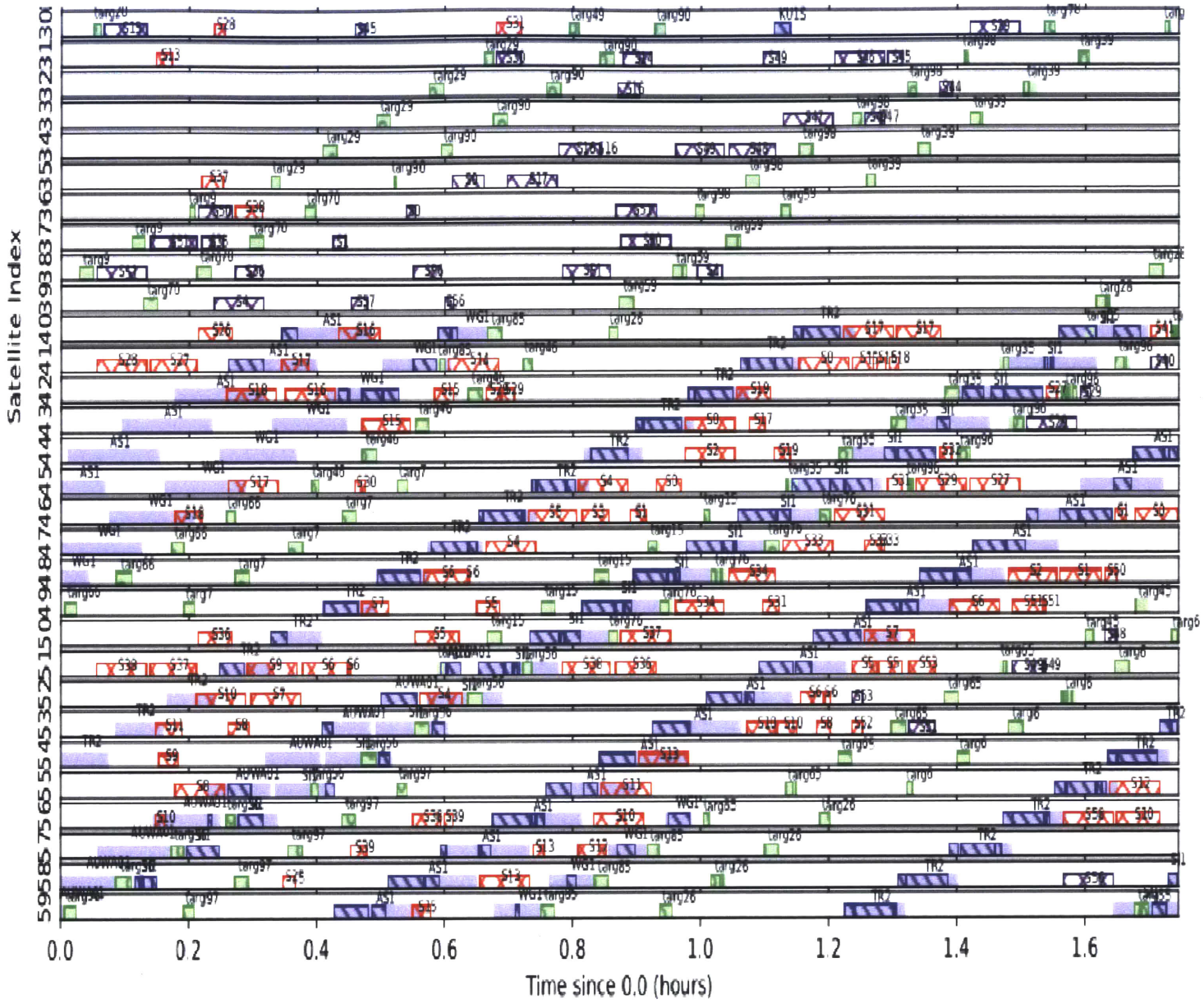


Figure A-7: 60 satellite: Full Schedule for 60 deg inclination constellation. (satellites 31-60)

Appendix B

Additional Properties, Methods, and Pseudocode

B.1 Object Properties and Methods

This section provides additional details on the key methods and properties used in the pseudocode blocks and descriptions in the thesis. The use of an appendix is intended to provide a reference for the interested reader that wants to understand more of the details than just the higher level points presented in the thesis. Properties will come after each description, then methods will be listed. The notation distinction is defined in the Symbols list.

- *Agent* object, shown in pseudocode as: I.
  - Consensus Dictionaries: The following three properties are what all agents need to agree on during a round of CMF to achieve consensus. Consensus is marked as achieved for a single agent when it has communicated with all of its neighbors and there are no changes. Once an agent hears that all agents have achieved consensus, it moves on to the next phase of the algorithm. Note: instead of referring to all three dictionaries, they will be collectively referred to as the consensus dictionaries. These dictionaries are used instead of just indexing the information because different agents

will create new tasks simultaneously that have the same unique ID, but may have different indices depending on the order of operations in the decentralized setting.

  * $I.bids(J.id) \in \mathbb{R}^{N_t}$: dictionary of latest bid information that has the $Task$ bid at the corresponding $Task$ ID. All values in the dictionary are initialized to 0. All keys are $Task$ IDs, denoted in pseudocode as $J.id$. Note: in the CBBA papers, this property serves the same purpose as the variable: $\mathbf{y}_i(t)$ [10, 61].
  * $I.winners(J.id) \in \mathbb{N}^{N_t}$: dictionary of latest winner information that has the $Agent$ ID corresponding to the $Task$ ID. All values in the dictionary are initialized to $null$. All keys are $Task$ IDs, denoted in pseudocode as $J.id$. Note: the domain is $\mathbb{N}$ because each unique ID string can be translated to a unique integer (and vice versa) based on the encoding used. Note: in the CBBA papers, this property serves the same purpose as the variable: $\mathbf{z}_i(t)$ [10, 61].
  * $I.inds(J.id) \in \mathbb{S}^{N_t}$: dictionary of latest planned timesteps, stored as indices in the discrete planning domain, that has the set of indicies corresponding to the $Task$ ID. All values in the dictionary are initialized to $\emptyset$. All keys are $Task$ IDs, denoted in pseudocode as $J.id$. Note: the domain of $\mathbb{S}$ means that each value of the dictionary is a unique set.

- $I.fd(t)$: property that holds the flow preference state of this agent. This property is calculated after each bundle phase is completed.
- $I.tmpAcc(J.id) \in \mathbb{R}^{N_t}$: dictionary that holds the accumulated value that a $Task$ accrues during the bundle phase. This addition is only required because of task forking for multibid tasks. If a task it not forkable or singlebid only, then this accumulator is not used. When it is used, each $ExecutableTask$ that is added to the bundle accumulates value in this dictionary, which is then used to check at the end of the bundle phase

if J has accumulated enough value to beat the current winning bid in $I.bids(J.id)$. If it has, then the winner is set to I and the $ExecutableTasks$ of J are kept on the bundle; otherwise, all the $ExecutableTasks$ associated with J are removed from the bundle. Then the bundle is rebuilt, with the exclusion of evaluating any $ExecutableTasks$ associated with J.

- checkMultibidTasks$_I()$: this method is run right before the end of each bundle phase and is required because of task forking. See appendix B.2 for more details.
- cmfBundlePhase$_I()$: bundle phase for CMF which modifies the following: $\mathbf{b_i}$, $\mathbf{p_i}$, and the consensus dictionaries. Has no inputs or return values. The pseudocode for this procedure is included in algorithm 6. Similar to CCBBA bundle phase, except for modifications required to support task forking and outflow coupling.
- updateBundleCache$_I()$: this method resets all satellite states (data, energy, and activity timeline, which are $d_i(t), e_i(t)$, and $a_i(t)$, respectively), then computes them based on everything in the current bundle, $\mathbf{b_i}$. Has no inputs or return values. This function is run at the start of each bundle phase because tasks may be removed from the bundle during the consensus phase, so the state needs to be repaired to reflect the current bundle when starting a new bundle phase.
- findBestAssignment$_I(J_e)$: this method returns three values: 1) $inds$, which are the indices of execution (must be sequential) that provide the most value for J_e, 2) c_{ij}, which is the value of J_e executed at $inds$, and 3) J_C, which is the flow-coupled task (must be a downlink and could be $null$ if no coupled task needed).
- forkTask$_I(J_e)$: the pseudocode for this algorithm is shown in algorithm 1.
- getOptimisticScore$_I(J_e, inds, c_{ij})$: this method is required because of task forking. Since there could be many $ExecutableTasks$ associated with a single $Task$ and the consensus dictionaries achieve consensus on the $Tasks$,

a mechanism is needed to allow a small portion of a $Task$ to be added to the bundle when the cumulative value of executing all $ExecutableTasks$ could beat the current bid associated with the $Task$. This method returns the best case value that the $Agent$ can achieve on $Task$ based on the current bundle and $inds$ selected for execution of J_e.

- createExecutableTasksFromInds$_I(newTaskIndsList, J)$: Creates new $ExecutableTasks$ based off the indicies provided in the list and the original $Task$ (J) that the $ExecutableTasks$ should point back to.
- getMutualDependentTask$_I(J)$: this method returns J_{MD}, which is the $Task$ that is mutually dependent with J. This is used to check and enforce mutual dependence instead of the more general activity matrices used in CCBBA. This is done because for max flow problems there will only be mutual dependence between two tasks at a time (the transmitter and the receiver). For the satellite context, this is represents a mutual dependence between every TX with a matched RX (and vice-versa).
- updateStates$_I(J_e)$: this method updates all satellite states based on the addition of a single task (J_e) to the bundle. It does this from the executed timepoints and activity type, $J_e.E_{inds}$ and $J_e.type$, respectively. This method also considers time-adjacent $ExecutableTasks$ to determine if a transition time is required or not. Note: transition time between $ExecutableTasks$ is only required when the parent $Task$ has a different $node\_id$ value because that means that the $Agent$ must slew to point at a different object.
- updateDistanceToDlnk$_I()$: Returns the distance to downlink value based on the satellites current bundle and knowledge of other satellites' distance to downlink. As explained in section 3.2.4, the distance to downlink value is used to calculate $I.fd(t)$ at each time step.

- $Task$ object, shown in pseudocode as : J.

- $J.node\_id \in \mathbb{N}$: this is the unique string associated with the physical location the satellite has to point to when executing the task. The correspondence of node type to task type is as follows: observation $\leftarrow$ target ID, downlink $\leftarrow$ ground station ID, TX $\leftarrow$ reception satellite ID, RX $\leftarrow$ transmission satellite ID.
- No methods. Tasks are static objects that are fixed once created. This is one of the major distinctions between $Tasks$ and $ExecutableTasks$.

- $ExecutableTask$ object, shown in pseudocode as: J_e.
  - $J_e.P_{inds} \in \mathbb{S}^{\mathbb{N}}$: Set of integers indicating the possible timesteps in the planning horizon this $ExecutableTask$ can be executed.
  - $J_e.E_{inds} \in \mathbb{S}^{\mathbb{N}}$: Set of integers indicating the timesteps in the planning horizon this $ExecutableTask$ is planned for execution.
  - $J_e.multibid \in \{true, false\}$: boolean property stores whether or not an $ExecutableTask$ is multibid or not. If $true$, that means that multiple agents can bid on the task and only the highest bidding agent can execute the task. In the default use case, this occurs with downlinks at the same ground station at the time time and in multibid transmission crosslinks. If $false$, it means that only one agent can bid on the task, so there is no need to update the consensus dictionaries since no other agents need to know about the execution of this task (with the exception of crosslink tasks, which always update the consensus dictionaries because they need to couple with their mutually dependent task).
  - $J_e.type \in \{xlnk, obs, dlnk\}$: property that stores the type of satellite activity.
  - $J_e.direction \in \{tx, rx\}$: denotes the flow direction of J_e. This is completely determined by $J_e.type$, with observations and reception crosslinks being rx, while downlinks and transmission crosslinks are tx.
  - $J_e.task \in \mathcal{J}$: returns the $Task$ that J_e corresponds to. Each J_e only has

one $Task$ as its parent; however, each $Task$ can have many $ExecutableTasks$ because of forking.

- updatePostFork$_{J_e}(R_{inds}, newTasks)$: This method removes R_{inds} from J_e's possible timesteps (P_{inds}) because those timesteps are allocated to the $newTasks$. This method also links the $newTasks$ to eachother as "forked siblings" and links them to J_e as their "forked parent". This creates a tree structure that can be used to quickly construct information about the portion of the original $Task$ scheduled for execution.
- updateExecTimeAndValue$_{J_e}(inds_B, c_{ij}^{best})$: this method modifies the following properties of J_e: $E_{inds} \leftarrow inds_B$ and $value \leftarrow c_{ij}^{best}$.

Some general functions referenced in the pseudocode.

- split(x): Splits the set or list stored in x based on consecutive values. Returns separate sets or lists of consecutive values (assumes integers in the input set or list).
- sort(x): Sorts x in ascending value.

B.2 CMF Full Bundle Phase Pseudocode

Because the bundle phase modifications alter the flow of the CCBBA bundle phase, the entire bundle phase pseudocode is provided here. Task forking allows $ExecutableTasks$ to be split up for a single $Task$, which provides the ability to increase value of a schedule. However, since the schedule consensus is at the $Task$ level, there needs to be a mechanism to validate that all multibid $Tasks$ that have $ExecutableTasks$ on the bundle were actually the highest bidder cumulatively. The function
GetOptimisticScore$_I(J_e, inds, c_{ij})$ allows for adding forked tasks to the bundle when their accumulated value could possibly beat the current bid, but is not guaranteed to beat the current bid. GetOptimisticScore$_I(J_e, inds, c_{ij})$ returns c_{ij} if $J_e.multibid = false$. On line 40, the function CheckMultibidTasks$_I()$ is used to ensure that only

the actual highest bidder gets to keep $ExecutableTasks$ associated with a $Task$ on its bundle.

The CheckMultibidTasks$_I$() function will remove all multibid tasks that are not actually the highest total bid by comparing $I.tmpAcc(J.id)$ to $I.bids(J.id)$. Once these are removed, they are prevented from being bid on and the bundle-phase is repeated until all $ExecutableTasks$ are valid. In all test cases so far, this recursion is only needed once per bundle phase, but it could theoretically go on multiple times if there are enough $ExecutableTasks$. CheckMultibidTasks$_I$() requires no additional communication since the information required is already stored in the consensus dictionaries build from previous consensus phases.

Note that when a value set to *null* is evaluated in an "if" statement, it will evaluate to $false$. This means the bundle phase will proceed to CheckMultibidTasks$_I$() if there is no "best" $ExecutableTask$ ($J_e^{best} = null$), which means that no remaining $ExecutableTask$ provided additional value.

The consensus phase pseudocode is not provided because the overall consensus phase execution and information agreement protocols are the same as CCBBA, except that additional information is being shared and time-matching is executed at the end of the consensus phase, but the time-matching only impacts the next round and doesn't impact the execution order of the consensus phase. Also note that Multibid tasks and crosslinks ($xlnks$) are the only tasks that are shared during the consensus phase.

Algorithm 6 CMF Bundle Phase

1: **procedure** CMFBUNDLEPHASE$_I$
2: $unchanged \leftarrow true$
3: updateBundleCache$_I()$
4: **while** $|\mathbf{b_i}| < L_t$ **do**
5: $J_e^{best} \leftarrow null$ ▷ Best scoring task
6: $c_{ij}^{best} \leftarrow 0$
7: $J_e^{coupled} \leftarrow null$ ▷ Flow-Coupled task for J_e^{best}
8: $inds_B \leftarrow null$
9: **for** J_e **in** $\mathcal{J}_i^{exec}$ **do**
10: $J \leftarrow J_e.task$
11: $canBid_{ij} \leftarrow true$
12: **if** $J_e.type = xlnk$ **then** ▷ only xlnks have mutual dependence
13: $J_{MD} \leftarrow$ getMutualDependentTask$_I(J)$
14: $canBid_{ij} \leftarrow (w_{ij} > 0) \vee (I.bids(J_{MD}.id) > 0)$
15: **if** $canBid_{ij} \wedge J_e \notin \mathbf{b_i}$ **then**
16: $inds, c_{ij}, J_C \leftarrow$ FindBestAssignment$_I(J_e)$
17: $\hat{c}_{ij} \leftarrow$ GetOptimisticScore$_I(J_e, inds, c_{ij})$
18: **if** $(\hat{c}_{ij} > I.bids(J.id)) \wedge (c_{ij} > c_{ij}^{best})$ **then**
19: $J_e^{best} \leftarrow J_e$
20: $c_{ij}^{best} \leftarrow c_{ij}$
21: $J_e^{coupled} \leftarrow J_C$
22: $inds_B \leftarrow inds$
23: **if** J_e^{best} **then**
24: $t_B \leftarrow inds_B(0)$
25: $\mathbf{b_i} \leftarrow \mathbf{b_i} \oplus_{end} \{J_e^{best}\}$
26: $\mathbf{p_i} \leftarrow \mathbf{p_i} \oplus_{t_B} \{J_e^{best}\}$
27: UpdateExecTimeAndValue$_{J_e^{best}}(inds_B, c_{ij}^{best})$
28: UpdateStates$_I(J_e^{best})$ ▷ updates $d_i(t)$,$e_i(t)$,$a_i(t)$
29: **if** $J_e^{best}.multibid$ **then**
30: $I.tmpAcc(J_e^{best}.task.id) \leftarrow I.tmpAcc(J_e^{best}.task.id) + J_e^{best}.value$
31: **else**
32: **if** $J_e{}^{best}.type = xlnk$ **then**
33: $I.inds(J_e{}^{best}.task.id) \leftarrow I.inds(J_e{}^{best}.task.id) \bigcup J_e^{best}.E_{inds}$
34: $I.bids(J_e^{best}.task.id) \leftarrow J_e^{best}.value$
35: $I.winners(J_e^{best}.task.id) \leftarrow I.id$
36: ForkTask$_I(J_e{}^{best})$ ▷ See alg. 1
37: $unchanged \leftarrow false$
38: **else** ▷ No more valid tasks to add
39: **Break** ▷ break out of while loop, goto line 40
40: CheckMultibidTasks$_I()$
41: **return** *unchanged*

Bibliography

[1] Johannes Aldinger and Johannes Löhr. Planning for Agile Earth Observation Satellites. *ICAPS Workshop on Planning in Continuous Domains*, pages 9–17, 2013.

[2] Mehdi Alighanbari. Task assignment algorithms for teams of UAVs in dynamic environments. page 118, 2004.

[3] M. Bernardine Dias, Robert Zlot, Nidhi Kalra, and Anthony Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, 2006.

[4] Dimitri P. Bertsekas. Auction Algorithms for Network Flow Problems. Technical report, 1992.

[5] Nicola Bianchessi, Jean François Cordeau, Jacques Desrosiers, Gilbert Laporte, and Vincent Raymond. A heuristic for the multi-satellite, multi-orbit and multi-user management of Earth observation satellites. *European Journal of Operational Research*, 177(2):750–762, 2007.

[6] Kerri Cahoy, Peter Grenfell, Angela Crews, Michael Long, Paul Serra, Anh Nguyen, Riley Fitzgerald, Christian Haughwout, Rodrigo Diez, Alexa Aguilar, John Conklin, Cadence Payne, Joseph Kusters, Chloe Sackier, Mia LaRocca, and Laura Yenchesky. The CubeSat Laser Infrared CrosslinK Mission (CLICK). Technical report, CubeSat Developers Workshop, 2019.

[7] X. Cai, D. Sha, and C. K. Wong. Time-varying universal maximum flow problems. *Mathematical and Computer Modelling*, 33(4-5):407–430, 2001.

[8] Michael Cashmore, Andrew Coles, Bence Cserna, Erez Karpas, Daniele Magazzeni, and Wheeler Ruml. Temporal planning while the clock ticks. *Proceedings International Conference on Automated Planning and Scheduling, ICAPS*, 2018-June:39–46, 2018.

[9] Steve Chien, Gregg Rabideau, Russell Knight, Robert Sherwood, B. Engelhardt, D. Mutz, Tara Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, and D. Tran. ASPEN: Automated Planning and Scheduling for Space Mission Operations. *SpaceOps*, (December 2000):1–10, 2000.

[10] Han Lim Choi, Luc Brunet, and Jonathan P. How. Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics*, 25(4):912–926, 2009.

[11] Emily Clements, Raichelle Aniceto, Derek Barnes, David Caplan, James Clark, Iñigo del Portillo, Christian Haughwout, Maxim Khatsenko, Ryan Kingsbury, Myron Lee, Rachel Morgan, Jonathan Twichell, Kathleen Riesing, Hyosang Yoon, Caleb Ziegler, and Kerri Cahoy. Nanosatellite optical downlink experiment: design, simulation, and prototyping. *Optical Engineering*, 55(11):111610, 2016.

[12] Frédéric Cristini. Satellite networks: Solutions against emerging space threats. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, 18(PART 1):380–385, 2010.

[13] CubeSat. 6U CubeSat Design Specification Rev. 1.0. Technical report, 2018.

[14] DARPA. Broad Agency Announcement Blackjack Pit Boss Tactical Technology Office. Technical report, 2019.

[15] Austin J. Dionne, Jordan T. Thayer, and Wheeler Ruml. Deadline-aware search using on-line measures of behavior. In *Proceedings of the 4th Annual Symposium on Combinatorial Search, SoCS 2011*, pages 39–46, 2011.

[16] Jonathan M. Dyer and Jim McClelland. Paradigm Change in Earth Observation - Skybox Imaging and SkySat-1. In *Proceedings of the 12th Reinventing Space Conference*, pages 69–89, 2017.

[17] Jeff Foust. Amazon-Lockheed venture casts shadow on ground station startups, 2018.

[18] Brian P. Gerkey and Maja J. Matarić. Sold!: Auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768, 2002.

[19] Fred Glover. Tabu Search - Part 1. *ORSA Journal on Computing*, 1(3):135–206, 1989.

[20] Fred Glover. Tabu Search - Part 2. *OSRA Journal on Computing*, 2(1):1–97, 1990.

[21] Andrew Goldberg and Robert Tarjan. A New Approach to the Maximum-Flow Problem. *Journal of the ACM*, 35(4):921–940, 1988.

[22] Peter Grenfell, Alexa Aguilar, Kerri Cahoy, and Michael Long. SSC18-WKI-01 for Small Satellite Laser Communications. In *32nd Annual AIAA/USU Conference on Small Satellites*, pages 1–7, 2018.

[23] Caleb Henry. SpaceX launches 60 Starlink satellites, begins constellation buildout - SpaceNews.com, 2019.

[24] Omid S. Jahromi. *Distributed Algorithms.* Morgan Kaufmann Publishers, Inc, San Francisco, CA, USA, 2007.

[25] Sushant Jain, Kevin Fall, and Rabin Patra. [2004] Routing in a Delay Tolerant Networking.pdf. In *SIGCOMM' 04*, 2004.

[26] Luke Johnson, Han Lim Choi, Sameera Ponda, and Jonathan P. How. Allowing non-submodular score functions in distributed task allocation. *Proceedings of the IEEE Conference on Decision and Control*, (1):4702–4708, 2012.

[27] Luke B. Johnson, Han Lim Choi, Ponda Sameera, and Jonathan P. How. Decentralized task allocation using local information consistency assumptions. *Journal of Aerospace Information Systems*, 14(2):103–122, 2017.

[28] Luke B. Johnson, Sameera S. Ponda, Han Lim Choi, and Jonathan P. How. Asynchronous decentralized task allocation for dynamic environments. *AIAA Infotech at Aerospace Conference and Exhibit 2011*, 2011.

[29] Andrew K. Kennedy. *Resource Optimization Algorithms for an Automated Coordinated CubeSat Constellation.* Master's thesis, Massachusetts Institute of Technology, 2015.

[30] Andrew Kitrell Kennedy. *Planning and Scheduling for Earth-Observing Small Satellite Constellations.* PhD thesis, Massachusetts Institute of Technology, 2018.

[31] S Kirkpatrick, C D Gelatt, and M P Vecchi. Optimization by Simulated Annealing. *S*, 220(4598):671–680, 2007.

[32] Leslie Lamport. Time, Clocks, and the Ordering of Events. *Comm ACM*, 21(7):558–565, 1978.

[33] Elizabeth Mabrouk. What are SmallSats and CubeSats?, 2015.

[34] Daniel Mandl, Gary Crum, Vuong Ly, Matthew Handy, Karl F Huemmrich, Lawrence Ong, Ben Holt, and Rishabh Maharaja. Hyperspectral Cubesat Constellation for Natural Hazard Response (Follow-on). In *Annual AIAA/USU Conference on Small Satellites*, pages SSC16–XII–02, 2016.

[35] MMA-Design. HaWK 17AB36, 2018.

[36] Philippe Monmousseau. Scheduling of a Constellation of Satellites: Improving a Simulated Annealing Model by Creating a Mixed-Integer Linear Model. 2015.

[37] K. Muthuchelian, N. Nedunchezhian, and G. Kulandaivelu. Small Spacecraft Technology State of the Art. Technical Report 4, 1994.

[38] Sreeja Nag, Alan S. Li, and James H. Merrick. Scheduling algorithms for rapid imaging using agile Cubesat constellations. *Advances in Space Research*, 61(3):891–913, 2018.

[39] Nano-avionics. CubeSat Propulsion "EPSS" – Green Chemical Propulsion System, 2019.

[40] NanoAvionics. NanoAvionics 6U satellite bus M6P, 2019.

[41] NASA. Near Earth Network (NEN) Users' Guide, 453-NENUG, 2016.

[42] Noam Nisan and Amir Ronen. Algorithmic Mechanism Design Algorithmic Mechanism Design contact. *Journal of Economic Literature*, pages 0–34, 1999.

[43] Scott Palo, Darren O'Connor, Elizabeth DeVito, Rick Kohnert, Gary Crum, and Serhat Altunc. Expanding CubeSat Capabilities with a Low Cost Transceiver. *AIAA/USU Conference on Small Satellites*, 2014.

[44] Cadence Payne, Angie Crews, Paul Serra, Kerri Cahoy, Alexa Aguilar, Peter Grenfell, and Haeyoung Choi. Laser Crosslink Atmospheric Sounder to Investigate the Effects of Deep Convection on Ozone, Massachusetts Institute of Technology , Depart. In *32 Annual AIAA/USU Conference on Small Satellites*, 2018.

[45] Thuy Lien Pham, Ivan Lavallee, Marc Bui, and Si Hoang Do. A distributed algorithm for the maximum flow problem. In *ISPDC 2005: 4th International Symposium on Parallel and Distributed Computing*, volume 2005, pages 131–138, 2005.

[46] Sameera S. Ponda, Luke B. Johnson, and Jonathan P. How. Distributed chance-constrained task allocation for autonomous multi-agent teams. Technical report, 2012.

[47] J. Puig-Suari, C. Turner, and W. Ahlgren. Development of the standard CubeSat deployer and a CubeSat class picosatellite. In *IEEE Aerospace Conference Proceedings*, volume 1, pages 1347–1353, 2001.

[48] Gregg Rabideau, Russell Knight, Steve Chien, Alex Fukunaga, and Anita Govindjee. Iterative Repair Planning for Spacecraft operations using the ASPEN system. *Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space, ISAIRAS '99*, 440(May):99–106, 1999.

[49] Michael Ricard and Stephan Kolitz. The ADEPT Framework for Intelligent Autonomy. In *Intelligent Systems for Aeronautics*, number May, pages 13–17, 2002.

[50] Kathleen Riesing, Hyosang Yoon, and Kerri Cahoy. A portable optical ground station for low-earth orbit satellite communications. *2017 IEEE International Conference on Space Optical Systems and Applications, ICSOS 2017*, (Llcd):108–114, 2018.

[51] Ricardo Rios-Olmo and Martin Miller. Planet's Open Water Imaging-Geo-Accuracy Assessment. *AIAA/USU Conference on Small Satellites*, (787), 2017.

[52] Christian Rodriguez, Henric Boiardt, and Sasan Bolooki. CubeSat to commercial intersatellite communications: Past, present and future. *IEEE Aerospace Conference Proceedings*, 2016-June:1–15, 2016.

[53] Scott H. Schaire, Harry Shaw, Serhat Altunc, George Bussey, Peter Celeste, Obadiah Kegege, Yen Wong, Yuwen Zhang, Chitra Patel, David Raphael, Jacob Burke, La Vida Cooper, James Schier, William Horne, and David Pierce. NASA near earth network (NEN) and space network (SN) CubeaSat Communications. In *SpaceOps 2016 Conference*, 2016.

[54] Thomas Schetter, Mark Campbell, and Derek Surka. Multiple agent-based autonomy for satellite constellations. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1882:151–165, 2000.

[55] SSTL. SGR-10 – Space GPS Receiver. Technical report, Surrey Satellite Technologies Ltd, 2013.

[56] Xueyuan Su, Sammy Chan, and Gang Peng. Auction in multi-path multi-hop routing. *IEEE Communications Letters*, 13(2):154–156, 2009.

[57] Michael Swartwout. The first one hundred cubesats: A statistical look. *Journal of Small Satellites*, 2(2):213–233, 2013.

[58] Martin N. Sweeting. Modern Small Satellites-Changing the Economics of Space. *Proceedings of the IEEE*, 106(3):343–361, 2018.

[59] S. R. Tsitas and J. Kingston. 6U CubeSat design for Earth observation with 6-5m GSD, five spectral bands and 14Mbps downlink. *Aeronautical Journal*, 114(1161):689–697, 2010.

[60] Sven Vries de and Rakesh V. Vohra. Combinatorial Auctions : A Survey. *Journal on Computing*, 15(3):284–309, 1998.

[61] Andrew K. Whitten, Han Lim Choi, Luke B. Johnson, and Jonathan P. How. Decentralized task allocation with coupled constraints in complex missions. *Proceedings of the American Control Conference*, pages 1642–1649, 2011.

[62] Zixuan Zheng, Jian Guo, and Eberhard Gill. Swarm satellite mission scheduling & planning using Hybrid Dynamic Mutation Genetic Algorithm. *Acta Astronautica*, 137:243–253, 2017.

[63] Zixuan Zheng, Jian Guo, and Eberhard Gill. Onboard autonomous mission replanning for multi-satellite system. *Acta Astronautica*, 145:28–43, 2018.

[64] Zixuan Zheng, Jian Guo, and Eberhard Gill. Distributed onboard mission planning for multi-satellite systems. *Aerospace Science and Technology*, 89:111–122, 2019.

[65] Di Zhou, Min Sheng, Xijun Wang, Chao Xu, Runzi Liu, and Jiandong Li. Mission aware contact plan design in resource-limited small satellite networks. *IEEE Transactions on Communications*, 65(6):2451–2466, 2017.