Crosslink Communication and Navigation Topology Identification for Distributed Systems Missions

Brin C. Harper
Maryann Rui
David D. Turner
Massachusetts Institute of Technology

February 16, 2022

I. Introduction

EMPLOYING constellations of satellites bring confers many advantages over using single satellites for many science and engineering applications. Distributed networks of satellites allow for broader coverage in in temporal, spatial, and spectral domains of both sensing and communication in temporal, spatial, and spectral domains. The satellite constellations that we consider in this project are NASA's Magnetospheric Multiscale (MMS) Mission, NASA's Afternoon Train, NASA's LunaNet constellation, and GPS-like Walker constellations. In this project, we consider of the A-Train and C-Train, the Magnetospheric Multiscale (MMS) Mission, the LunaNet constellation currently in development, and Walker-Delta Constellations for Global Positioning Systems (GPS). SeeSee Figure 1 for some illustrations of these examplesconstellations.

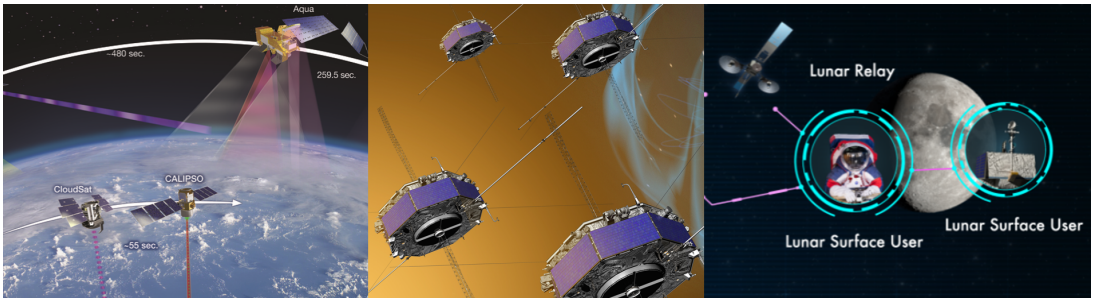

Fig. 1 Depictions of NASA satellite constellations examples, left to right: A-Train and C-Train, MMS, and LunaNet. Source: `nasa.gov`

In order to have a successfully working satellite constellation, we aim to address and enable several key capabilities: Several goals and capabilities we have for a successfully working satellite constellation include:

1) Information sharing and synchronization between satellites.
2) Having individual satellites inform all other satellites of a detected event.
3) Coordination of the constellation with a ground station through a designated base satellite.

The main question we must solve in realizing these goals is: *Given a satellite constellation, how can satellites efficiently route information along their inter-satellite communication links (ISL)?*

We can frame the problem in terms of message routing in graphs. We hereby interchangeably refer to satellites as nodes, and ISLs as crosslinks or edges. In light of our main question, we translate our goals into graph theoretic problems, which are depicted in Figure 2. The first goal– information sharing– can be seen as one-to-one or many-to-many communication, where a set of source nodes needs to relay its messages to a set of destination nodes. This includes the case where all nodes need to be updated on all other nodes' statuses. The second goal of propagating information from a single node to all other nodes is an example of one-to-many communication. Finally, having a single designated node collect all other nodes' messages to communicate with the ground station is a case of many-to-one communication. In all cases, we define an "efficient" routing specification as one that minimizes the total time to propagate all requisite messages from their sources to their destinations. See Figure 2

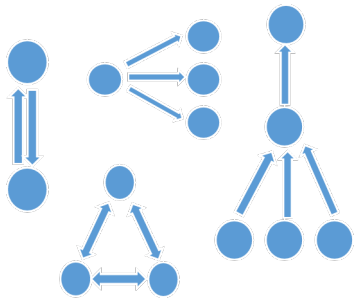

Fig. 2 Diagram of one-to-one, one-to-many, many-to-one, and many-to-many node communication in a graph.

Challenges. There are three key properties of this problem that make it challenging. First, for any given constellation, the relative positions of satellites change over the course of the orbit and sometimes occluding objects may cut the access between two satellites. Thus, we must route messages over a graph with dynamic connectivity profiles. Next, the rate of data transfer between two satellites depends on their relative distance and access profile at any given time. Finally, and probably the most significant challenge, is that each node can only communicate one way, to one other node at a given time. Therefore, even if you want a message to follow a given routea message is assigned to follow a given route, it may have to wait in a queues along at intermediate nodes before it can get sent through along because of channel sharing delays. So Thus, the actual message propagation time depends on all message paths selected, and the problem is not as straightforward as shortest path optimization.

The first challenge is resolved under the assumption that the time scale of any message passing session, which is on the order of seconds to a couple of minutes, is much smaller than the time scale of the changing access profiles of the constellation. This assumption holds better for some constellations, such as MMS, more than others such as LunaNet where there are more occlusions, i.e., the moon. See the plots in appendix V.D for some examples of how the constellations change over time starting at a given epoch generated from II.A.

The second and third challenges are specifically taken into account in the two algorithmic approaches to that we developed to solve the problem of optimal routing of messages between nodes, and which we present the problem that we present in this paper. We developed two approaches to solving the problem of optimal routing of messages between nodes:

1) Alternating minimization for routing and scheduling
2) Greedy near-Hamiltonian Cyclic cyclic Routing routing

The rest of the paper is organized as follows. In Section II.A we present the modeling and simulation of the satellite constellations of interest to which our crosslink communication algorithms are applied: NASA's LunaNet, NASA's Magnetospheric Multiscale (MMS) Mission, a GPS-like Walker constellation, and Nasa's NASA's Afternoon-Train Constellation. In Section III we present the approach and results of alternating minimization for message routing and scheduling. In Section IV we present the approach of near-Hamiltonian cyclic routing and its results. Finally, we conclude with a discussion in Section V comparing the two message routing algorithms and key takeaways.

II. Methods

A. Simulation and Data Collection

In order to represent each satellite constellation at a given epoch as a graph, we needed to simulate the relative dynamics of each satellite while keeping track of occultations and the changing slant range between individual satellites. We selected AGI's Systems Tool Kit (STK) for its wide range of analysis features, large database of two-line element sets (TLE), and ease of integration with MATLAB. Four kinds of constellations were generated and propagated in STK: NASA's LunaNet, NASA's Magnetospheric Multiscale Mission (MMS), a GPS-like Walker constellation, and NASA's Afternoon Constellation.

Each constellation was propagated over a certain time period, which varied for each constellation depending on the maximum orbital period. At each timestep, access periods and slant range between each pair of satellites were calculated. This data was used to construct an adjacency matrix A at each timestep, where $A_{i,j}$ is equal to the slant range between satellites i and j if there is an unobstructed line of sight between the satellites, and $A_{i,j} = 0$ otherwise. In theory, this could lead to confusion between the case where two satellites have an unobstructed line of sight but slant

range close to zero, and the case where two satellites have no line of sight, as both cases would be represented by zero elements in an adjacency matrix. However, we assumed that it was extremely unlikely for two satellites in the same constellation to be close enough that their slant range would be approximated as zero in MATLAB*. Furthermore, a near zero slant range would represent a conjunction between the satellites, which could render them unable to communicate.

As of January 2022, LunaNet (a conceptual lunar constellation) has not been deployed. Our simulated LunaNet constellation is based on the currently proposed example configuration: "Two relay orbiters phased 180° apart on the 12-hour frozen elliptical orbit with its line of apsides liberating over the North Pole, plus [t]wo relay orbiters phased 180° apart on the 12-hour frozen elliptical orbit with its line of apsides liberating over the South Pole, plus [o]ne relay orbiter on the 12-hour circular orbit around the equator" [1][2]. We added an additional spacecraft representing a LunaNet user in low lunar orbit (LLO). The LLO spacecraft's orbital parameters were based in part on the 2-h period mapping orbit of NASA's Lunar Reconnaissance Orbiter (LRO) [3]. These orbits were propagated over 1 day (86400 seconds), and adjacency matrices were constructed every 60 seconds.

To simulate the Magnetospheric Multiscale Mission (MMS), we used the TLEs for the four spacecraft from STK's satellite database. These were propagated over 1 year (31536000 seconds), and adjacency matrices were constructed every 42300 seconds.

Our GPS-like Walker Delta constellation is composed of 24 Earth-orbiting satellites in 6 equally spaced orbital planes. Each orbit is circular and has an altitude of 20,200 km. These orbits were propagated over 1 day (86400 seconds), and adjacency matrices were constructed every 1800 seconds.

To simulate the Afternoon Constellation, we used the TLEs for the A-Train satellites (OCO-2, GCOM-W1, Aqua, and Aura) and C-Train satellites (CALIPSO and CloudSat) from STK's satellite database. These orbits were propagated over 1 day (86400 seconds), and adjacency matrices were constructed every 60 seconds.

III. Alternating Minimization for Routing and Scheduling

The main idea of the Alternating Minimization for Routing and Scheduling approach is to realize that we have two coupled problems– one of routing and finding optimal message paths, and the other of scheduling and prioritizing which messages should be sent across given edges at any given time. Ideally, we would like to solve both problems simultaneously, but given that this problem is a mixed integer nonlinear problem, solving the general problem would require a prohibitive amount of computation. Instead, we alternate between find the best message route with given estimated message propagation and queue wait times along given paths with the *router* component, and then feed these message paths in to a *scheduler*, which will optimally schedule the message passing, and get an improved estimate on the edge travel times to re-run the router, and so forth. See Figure 3 for an illustration.

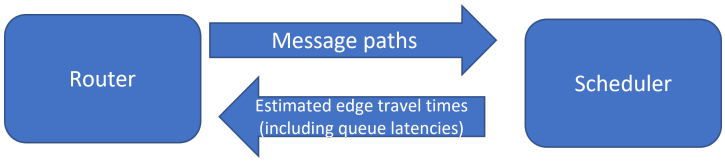

Fig. 3 Diagram of alternation between scheduler and router subroutines.

A. Scheduler Algorithm

The scheduling algorithm consists of extracting message passing actions (j, k, i), representing message i being sent from node j to node k, building an action graph with edges to enforce the message propagation order constraint, choosing a valid set of actions to simultaneously execute at a given step, and scheduling these actions for the constituent nodes involved in order to minimize the total message propagation time (MPT). From the schedule, we estimate the time it takes to send given messages across given edges in the graph, which we refer to as estimated edge-message propagation times (edge-MPTs). These estimates are fed into the router algorithm as a heuristic for selecting the best set of message paths to minimize the total MPT. See Figure 4 below for an example of the entire run of the scheduler algorithm.

*As of MATLAB 2021, the smallest positive normalized floating-point number that can be represented is 2^{-1022}. For this study, it is safe to assume that no pair of satellites in the same constellation will have a slant range anywhere near 2^{-1022} km.

Algorithm 1 Scheduler Subroutine

Input: $\{M_s : s \in S\}$ directed message paths, G_0 = original undirected graph of nodes with actual distances as edge weights, *travel_time*(j, k, i) = function to calculate travel time of a message of size m_i along edge $j \to k$.

Output: Schedule of actions $(j, k, i) := \{$send message i along edge $j \to k\}$ occur at any given time point, and the estimated edge-MPTs θ^i_{jk} for (j, k, i) tuples featured in $\{M_s : s \in S\}$.

1: Create action graph G_a with Message Propagation Order Constraint (MPOC) edges.
2: Iterate until no more visited actions remain:
3: **while** G_a has unvisited action nodes **do**
4: Create current fringe graph with No Time Overlap Constraint (NTOC) edges and heuristic action node weights (e.g., number of hops or estimated propagation time left for the given message).
5: Current selected action set $A \leftarrow$ Maximum Weight Independent Set (Max WIS) of the current fringe graph
6: Mark action nodes in A as visited in G_a
7: **end while**
8: Schedule the actions as early as possible for the constituent nodes, while maintaining their relative order from previous step.
9: $\theta^i_{jk} \leftarrow \tau^i_k - \tau^i_j$ ▷ Estimate message propagation times for (j, k, i) (edge, message) tuples featured in M_s.

1. Building the action graph.

We create the action graph G_a by extracting from the set of message paths all the individual actions that must be taken, and label them as (j, k, i) tuples, representing message i being passed from node j to k. These actions (j, k, i) make up the nodes of G_a. The directed edges are formed based on Message Propagation Order Constraints. There is an edge from (j, k, i) to (k, l, i) to indicate that node k needs to receive message i (from the node j preceding it in message i's path digraph) before it can pass message i forward to the next node l.

2. Creating the fringe graph and selecting the next batch of actions.

As long as there are actions nodes that have not yet been marked as visited (i.e., completed), we extract from G_a a set of fringe nodes. The fringe set is defined as the set of unvisited action nodes (j, k, i) in G_a which have no unvisited predecessors. This ensures that in the current step we are only considering actions that satisfy the MPOC constraint, i.e., that node j in the original graph will already have received message i. Next, create a fringe graph on these set of fringe nodes by adding edges representing the No Time Overlap Constraint (NTOC). Add an edge between two fringe action nodes $(j_1, k_1, i_1), (j_2, k_2, i_2)$ if and only if at least one node is active in both actions (e.g., $j_1 = k_2$).

We would like to select the maximum independent set (IS) of vertices on this fringe graph, and the fringe edges ensure that the max IS set of actions obeys the constraint that each node in the original graph can only be involved with one action at a time. However, motivated by Lemma 1, we also want to prioritize those actions involving messages that still have a long way to go before they reach their destination(s).

Thus, we weight each node (j, k, i) by a heuristic representing the time left for a message to propagate down its path. The simplest heuristic is to use the number of descendants in message i's subpath starting at the node k, plus 1, i.e., the number of hops left. We can also use some estimate of the remaining message propagation time left. Given these weights, we then select the next set of actions by choosing the maximum *weighted* independent set (WIS), and mark these actions as visited in G_a. Taking the max WIS allows us to balance prioritizing important actions and maximizing the number of actions that can be simultaneously run at a given time.

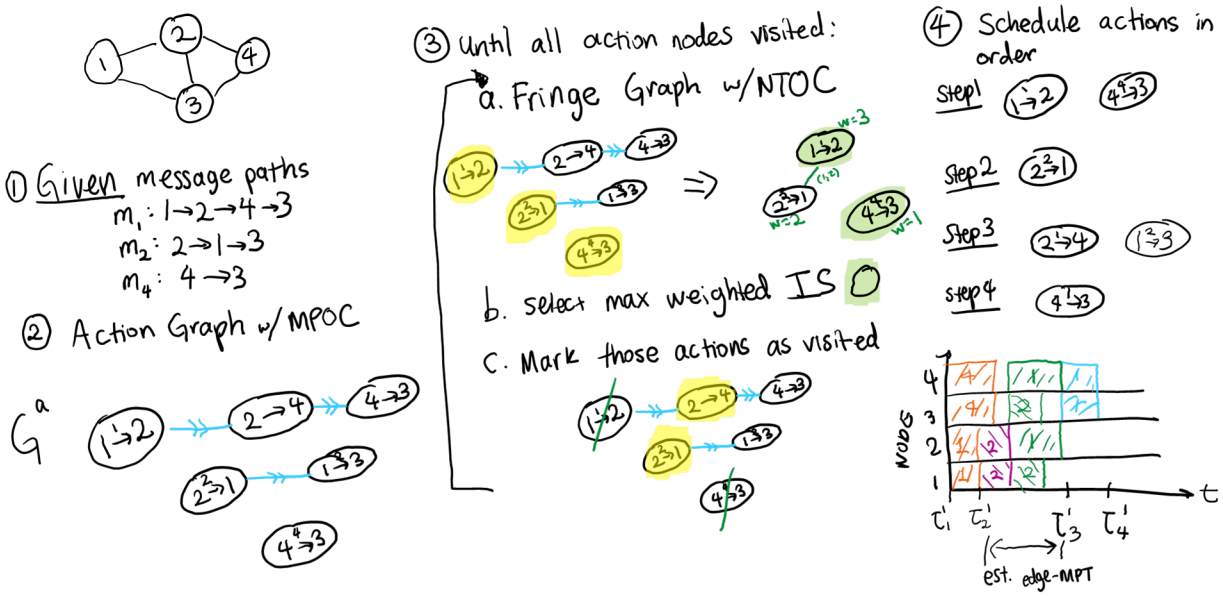

Fig. 4 An example run through of the scheduling algorithm: Given a set of message paths, construct the action graph, and until all actions are visited, create a fringe graph and select the max weighted set of actions. Finally, schedule the actions in the order selected and estimate edge-MPTs.

3. Motivation for prioritization heuristics in fringe graph.

A lemma on the prioritization of message transmission in a simplified setting motivates our heuristics of weighting action nodes by the number of hops their associated messages have remaining to complete their message paths. Suppose a given parent node 0 needs to transmit a message m_0, to each of its k children, given by the set $c(0) \subset [n]$. It takes $t_i j$ time to transmit m_0 from node 0 to node j, $j \in c(0)$. Once node j has received message m_0, it takes remaining time τ_j for the message to complete its subpath starting from node j (i.e., for m_0 to be further propagated down to the leaves of the subtree rooted at node j). The goal is to minimize the time it takes for m_0 to finish propagating down all its subpaths starting from node 0, subject to the constraint that node 0 can only transmit a message to one other node at a given time.

Lemma 1. *Given a node 0 with k children ($|c(0)| = k$), for each node j, let $\tau_j \geq 0$ be the time that it takes for the message m_0 to finish propagating through the sub-path starting at node j, and let $t_{0j} > 0$ be the time it takes to transmit m_0 from node 0 to child node $j \in c(0)$. Then the optimal order, in terms of minimizing the total message propagation time (MPT), in which to transmit messages to node 0's children is in the order $((1), (2), ..., (k))$ such that $\tau_{(1)} \geq \tau_{(2)} \geq ... \geq \tau_{(k)}$ where node (j) is the jth node in this ordering. That is, a parent node should broadcast its message first to the child nodes that has the longest remaining message propagation time. The total MPT of parent node 0 is*

$$\tau_0(((1), ..., (k))) = \max t_{0(1)} + \tau_{(1)}, t_{0(1)} + t_{0(2)} + \tau_{(2)}, ..., \sum_{j \leq k} t_{0(j)} + \tau_{(k)}. \quad (1)$$

See Figure 5. Note in particular the the order of nodes j to transmit to does not depend on how long the transmission time from 0 to j, but only node j's remaining MPT. The intuition is that the transmission time from node 0 to all of its children must be born in any case, and the goal is to ensure the longest τ_j's are initiated as earliest as possible to minimize the overall MPT.

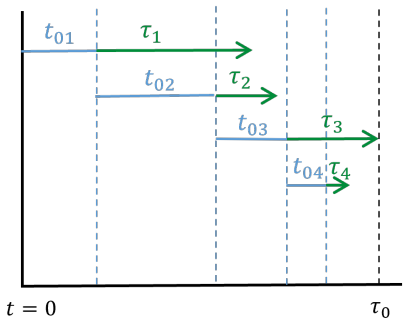

Fig. 5 (Lemma 1) Scheduling of message passing from node 0 to its children (in blue) and the subsequent propagation time from node i's children down each child's subtree (in green).

Proof. For ease of notation, suppose node 0's k children, $c(0) = [k]$, are ordered by decreasing remaining message propagation time (MPT), so that $\tau_1 \geq \tau_2 \geq .... \geq \tau_k$.

Actually first, suppose that node 0 has only 2 children $i \in \{1, 2\}$ with remaining MPTs $\tau_1 \geq \tau_2$.

It is clear that given any ordering of $c(i)$, node 0 should transmit to the nodes in this ordering one after another (with no wait time in between nodes), in order to minimize the total MPT. So our problem reduces to choosing an ordering $((1), (2)) = (1, 2)$ or $((1), (2)) = (2, 1)$ of node 0's children to minimize

$$\tau_0(((1), (2))) = \max\{t_{0(1)} + \tau_{(1)}, t_{0(1)} + t_{0(2)} + \tau_{(2)}\}$$

We claim that $(1, 2)$ is the optimal (MPT-minimizing) ordering. Suppose $\tau_1 = \tau_2$. Then both orderings produce the same τ_0 and so our claim holds trivially. Thus, suppose that $\tau_1 > \tau_2$. Let $A := \tau_0(1, 2)$ and $B := \tau_0(2, 1)$, and we will show that $A < B$ when $\tau_1 \geq \tau_2$. There are two possible values for A and two for B:

$$A = t_{01} + \tau_1, \text{ or } A = t_{01} + t_{02} + \tau_2$$
$$B = t_{02} + \tau_2, \text{ or } B = t_{02} + t_{01} + \tau_1.$$

However, $B = t_{02} + \tau_2$ implies that $\tau_2 \geq t_{01} + \tau_1$, which contradicts our premise that $\tau_1 > \tau_2$, since all time quantities are nonnegative. Thus it must be that $B = t_{02} + t_{01} + \tau_1$. Now if $A = t_{01} + \tau_1$, then $B - A = t_{02} > 0 \Rightarrow B > A$. And if instead, $A = t_{01} + t_{02} + \tau_2$, then $B - A = \tau_1 - \tau_2 > 0 \Rightarrow B > A$. Thus in all cases, it is $(1, 2)$ is the MPT-minimizing ordering, when $\tau_1 > \tau_2$.

Now consider the case of k children. We show that the nodes should be propagated to in order of decreasing τ_j by showing that we can always weakly decrease the total MPT by re-ordering the nodes in terms of decreasing τ_j. Suppose the nodes in $c(i)$ are ordered $(1, 2, ..., k)$ such that $\tau_1 \geq \tau_2 \geq ... \geq \tau_k$ does not hold. Then there are two consecutive nodes a, b, such that $a \in \{1, ..., k-1\}$, $b = a+1$, such that $\tau_a < \tau_b$. We claim that the ordering $O_2 = (1, 2, ..., b, a, ..., k)$ produces a smaller MPT than the original $O_1 = (1, 2, ..., a, b, ...k)$. In the expression for the total MPT, (1), note that the first $a - 1$ terms considered in the max operation are the same for O_1 as for O_2, as are the remaining $b + 1$th and onward terms. Thus the only way for the max to differ is in the ath and bth terms, $\sum_{j \leq a} t_{0j} + \tau_a$, $\sum_{j \leq b} t_{0j} + \tau_b$, which upon grouping common terms, reduces to $\sum_{j<a} t_{0j} + \max\{t_{0a} + \tau_a, t_{0a} + t_{0b} + \tau_b\}$. However, since $\tau_a < \tau_b$, our proof of the Lemma for $k = 2$ children shows that $\max\{t_{0b} + \tau_b, t_{0b} + t_{0a} + \tau_a\} < \max\{t_{0a} + \tau_a, t_{0a} + t_{0b} + \tau_b\}$, and so ordering $\tau_0(O_2) \leq \tau_0(O_1)$ and we can always weakly improve the total MPT by ensuring any two consecutive nodes are ordered in decreasing τ_j. Thus the optimal ordering to minimize the total MPT is achieved by that of decreasing remaining MPTs: $\tau_{(1)} \geq ... \geq \tau_{(k)}$.

Finally, the total MPT is the maximum time it takes for any of the children in $c(0)$ to finish propagating the message down their respective message sub-paths. The j transmission begins at $\sum_{l<j} t_{0l}$, it takes t_{0j} time to get transmit m_0 to j, and τ_j time for the message to propagate down the rest of its path starting at j. This leads to the expression (1) for the total MPT. □

This result motivates our heuristic for weighting the action nodes in the fringe graph in terms of the number of hops left, or estimated message propagation time left, for a message to complete its sub-path starting at the end node of the action. In order to minimize the total message propagation time, we try to prioritize actions passing messages that still have a long way to go to their destinations, while also trying to maximize the number of actions we can perform at a given time. Finding a maximum weighted independent set balances these two goals.

4. On selecting the maximum weighted independent set.

Note that finding the max IS or the max WIS of a graph is an NP-hard problem. To sidestep this problem of complexity, we instead estimate the max WIS using the best of two greedy algorithms that are analogous to common greedy approaches for the max (unweighted) IS problems [4]. The two main approaches are

1) WGMIN: Greedily selecting the most promising (here, determined by a heuristic value) node until we reach a maximal independent set (A maximal independent set is one that cannot include another node without violating the independent set property; Every maximum IS is maximal, but not vice versa.)
2) WGMAX: Greedily removing the least promising (again, determined by a heuristic value) nodes from the graph until no edges remain, so we are left with a (maximal) independent set.

The heuristic values for a given node v combine both the degree of the node $d(v)$ and its weight $w(v)$, and in each of the greedy algorithms are given by:

1) WGMIN: $h(v) = \frac{w(v)}{d(v)+1}$
2) WGMAX: $h(v) = \frac{w(v)}{d(v)(d(v)+1)}$

5. Scheduling of ordered actions.

Once we have an ordering of actions determined by the independent sets selected in each step of the scheduler algorithm, we create schedules of actions for each node in the original graph. Each action (j, k, i) involves two nodes j and k. Find the earliest time in which these two nodes are both free, but only after the nodes have executed all their previously assigned actions. That is, do not change the order of actions even if there is room in the schedule. This maintains the MPOC constraints. Keep track of the time at which each message arrives at its destination node, τ_k^i; especially the time of the last message to arrive, which is the total message propagation time.

We then estimate message propagation times along given nodes from the action schedule for action tuples (j, k, i) featured in M_s:

$$\theta_{jk}^i = \tau_k^i - \tau_j^i$$

This estimate incorporates the queueing delays experienced by a message along its route. However, note that we'll only have edge-message propagation time (edge-MPT) estimates for those action tuples in $\{M_s\}$, so some exploration may need to be done to get better estimates of queue latencies along other paths and edges. These latencies are only exact for the scheduled set of message paths $\{M_s\}$, but this holds at the fixed points of our algorithm, where the message paths selected produce exactly the estimated edge-MPTs through the scheduler.

B. Router Algorithm

Given our initial constellation graph along with estimated edge-message propagation times (edge-MPTs) along with a set of source and destination nodes, the routing algorithm returns a set of acyclic directed graphs representing the path each source node's message takes to reach all destination nodes. Note we use the term message "path" to refer to any acyclic directed graph with a source node and we are not restricted to 'linear' paths through the graph.

Algorithm 2 Router Subroutine

Input: G_{est} = undirected graph of nodes with estimated edge-MPTs θ_{jk}^i (sending message i from node j to k as edge weights, S = set of source nodes, T = set of common destination nodes,

Output: $\{M_s : s \in S\}$ = acyclic directed graphs representing node s's message routes for each $s \in S$.

1: **for** $s \in S$ **do**
2: $M_s \leftarrow$ directed shortest path tree from s to all nodes in T via Dijkstra's algorithm.
3: **end for**

C. Output of the alternating minimization algorithm

Our iterative scheduler-router algorithm allows us to explore different sets of message paths and try to improve on the total MPT. Let us define the "state" of the algorithm to be the set of message paths and the estimated edge-MPTs. Empirically, we find that the algorithm state always converges to a fixed point, at which the estimated edge-MPTs don't change in the next (and all future) iterations, and neither does the chosen set of message paths.

Note that the total number of possible sets of message paths, though combinatorially large, is finite. Further, the total message propagation times for any set of message paths is bounded (e.g., by the sum of times to send the largest message across every single edge in the graph), since the scheduler is trying to approximately minimize the total MPT and can do no worse than this upper bound. In fact edge-MPTs can only take a finite number of values (though again, combinatorially many depending on message paths and prioritization in queues). Thus our algorithm is stepping around in a finite state space and cannot diverge. Additionally, because our algorithm is deterministic on this finite state space, it can only cycle or converge on a fixed point. While cyclic behavior may or may not be a possibility, it has not been observed empirically. Figure 6 shows the MPT versus algorithm iteration and the normalized change in edge-MPT estimates per iteration. Here, the algorithm hits a fixed point at the 8th iteration, which happens to yield the lowest total MPT observed so far, but this minimality is not always the case.

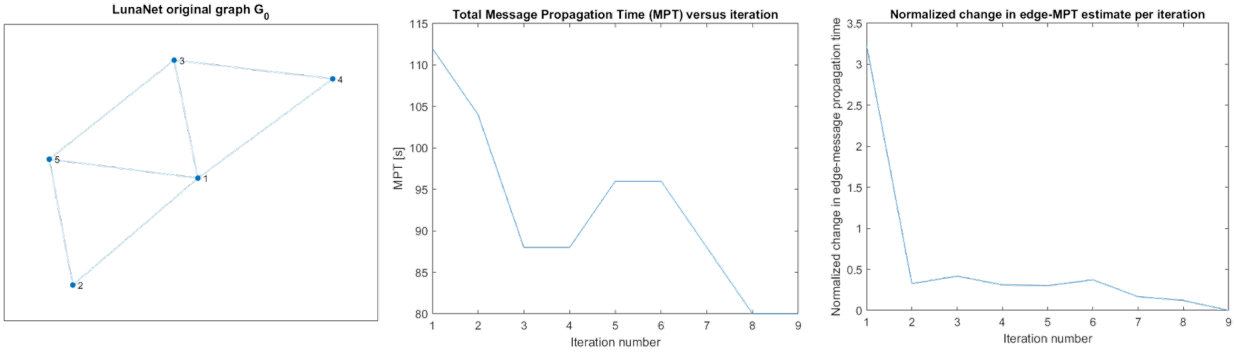

Fig. 6 Run of the router-scheduler algorithm for LunaNet at $T_0 = 0$. The plots from left to right show: (1) access graph of constellation (2) the total message propagation time versus iteration number, and (3) the normalized changes in the edge-MPT estimates per iteration.

Although we do have observe of the algorithm, there is no guarantee of optimality, i.e., that the algorithm wiill find the best message paths and scheduling of queues. As such, we take the output with the minimal MPT in the run of the algorithm, which is not necessarily the final output of the algorithm, as our approximation to the best message paths and scheduling.

Simulated Annealing Indeed, one of the benefits of the algorithm is that it gives us an automated way to explore different sets of message propagation paths while still trying to optimally schedule the actions. In order to increase this exploration, we also tried perturbing the estimated edge-MPTs with random noise (in the simulations below, we took Gaussian noise with a variance of approximately one-fourth the max estimated edge-MPT value. In doing so, we hope the router algorithm may explore nearby message paths that might be missed if we converge to only a single set of estimated edge-MPTs. Note that the algorithm will no longer converge to a fixed point since the state is always being perturbed by noise, but as before, we simply run the algorithm for a certain number of steps and take the step with the minimum MPT observed thus far. See Figure 7 for an example run of the algorithm using simulated annealing.

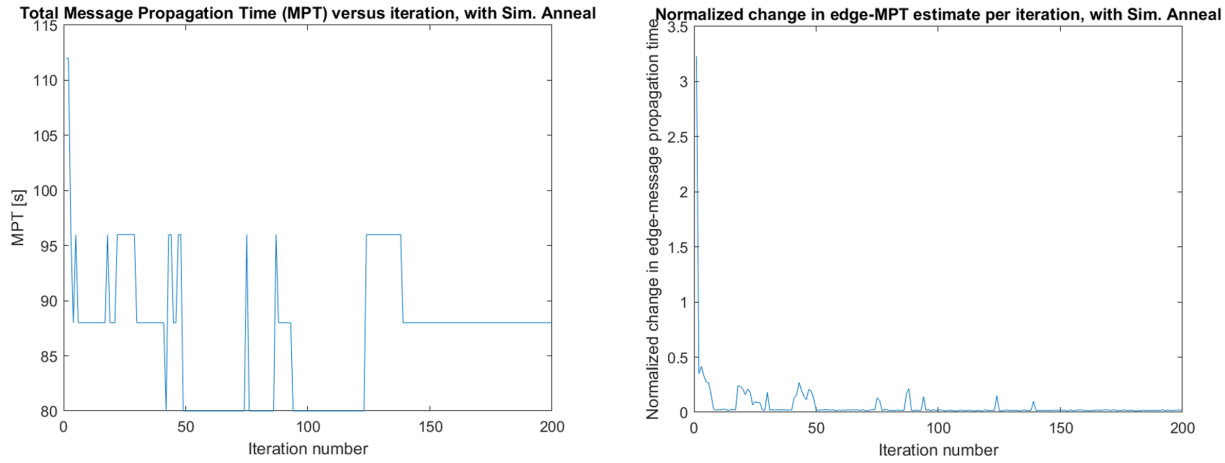

Fig. 7 Run of the router-scheduler algorithm for LunaNet at $T_0 = 0$ using simulated annealing. The left plot shows the total message propagation time versus iteration number, and the right plot shows the normalized changes in the edge-MPT estimates per iteration. Note that the algorithm will not converge on any fixed point because of the added noise.

D. Results

Here we present several example outputs of the algorithm, which consists of the set of message paths from source nodes to destination nodes, and the schedule of message propagation along the paths. We run the routing-scheduling algorithm on three of the four example constellations generated in Section II.A: A-Train C-Train, LunaNet, and MMS, assuming that each node in the constellation must share a 1kb message with every other node in the constellation. Figure 8 shows the graph of the A-Train C-Train constellation nodes with the time it takes to send a 1kb message between any two nodes that have access (i.e., share an edge), along with the best path for node 2's message to propagate to all the other nodes. The data rates used to determine message propagation times are determined by an exponential relation explained in III.D.1.

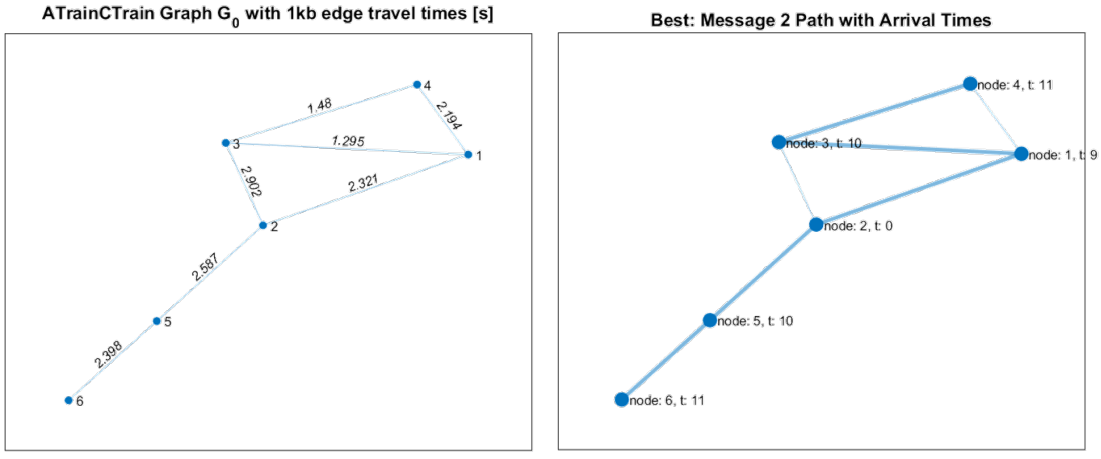

Fig. 8 A-Train C-Train graph at $T_0 = 0$ with 1kb message propagation times across edges in the left figure. In the right figure, the algorithm's output for the best path for node 2's message to propagate to all other nodes in the constellation.

In each of the first 5 panels of figures 9 and 10, we see the progress that the message from a given source node makes along its path at the time points $t = 5.42$s and $t = 42.03$s in the LunaNet constellation. The last panel shows the messages that are actively being sent within the graph at these two time points relative to the start of the message propagation.

Finally, Figure 11 show the best MPT output by the routing-scheduling algorithm for approximately the first 100 days of the ephemeris starting from the epoch ($T_0 =$) for MMS and 3.3 hours for the LunaNet constellation. We note the periodicity of about 84 hours in the MMS constellation and the variation of total MPT, though not quite periodic, in the LunaNet constellation. In future work, it is desirable to have a set of descriptive statistics that accounts for the variability of the MPT over the orbit of constellations.

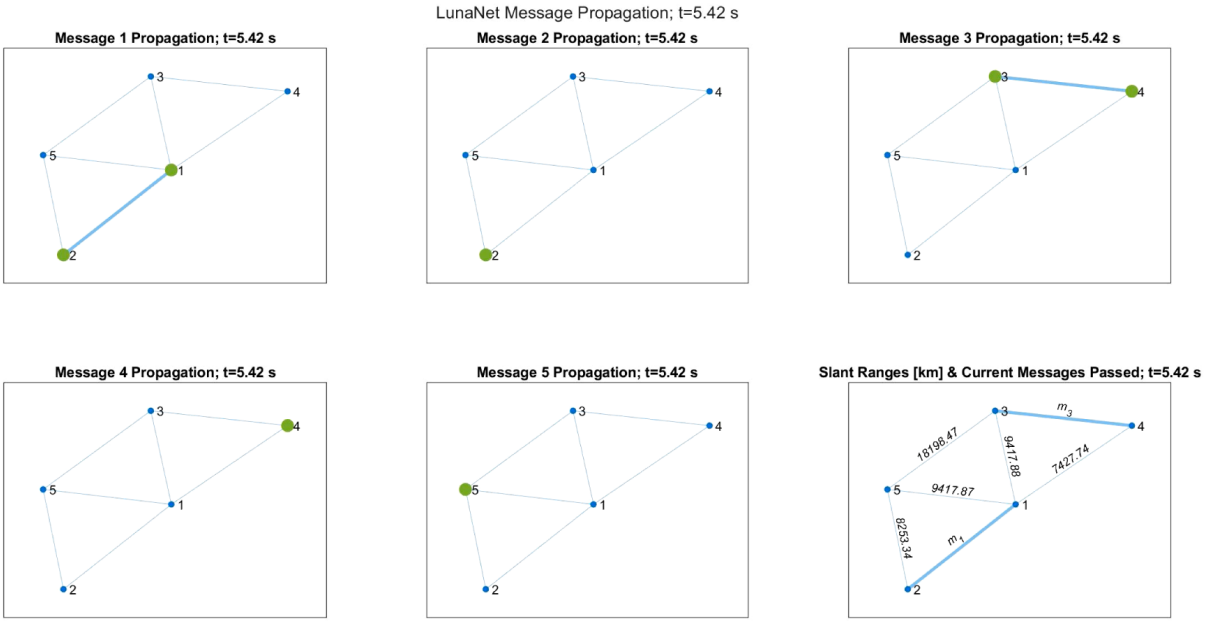

Fig. 9 Message propagation on the LunaNet constellation at $T_0 = 0$, for 1kb messages originating from all nodes arriving to all other nodes. The first 5 panels show each message's propagation progress. The last panel shows the current message being passed at $t = 5.42$s after the start of propagation, along with the slant ranges between nodes.

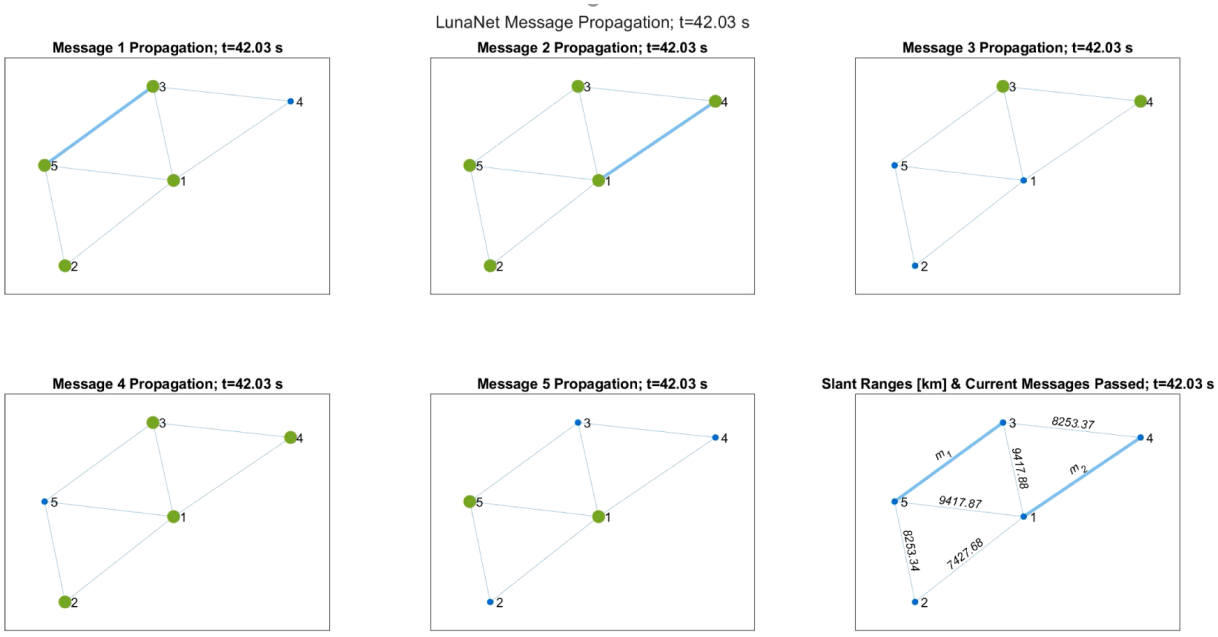

Fig. 10 Message propagation on the LunaNet constellation at $T_0 = 0$, for 1kb messages originating from all nodes arriving to all other nodes. The first 5 panels show each message's propagation progress. The last panel shows the current message being passed at $t = 42.03$s after the start of propagation, along with the slant ranges between nodes.

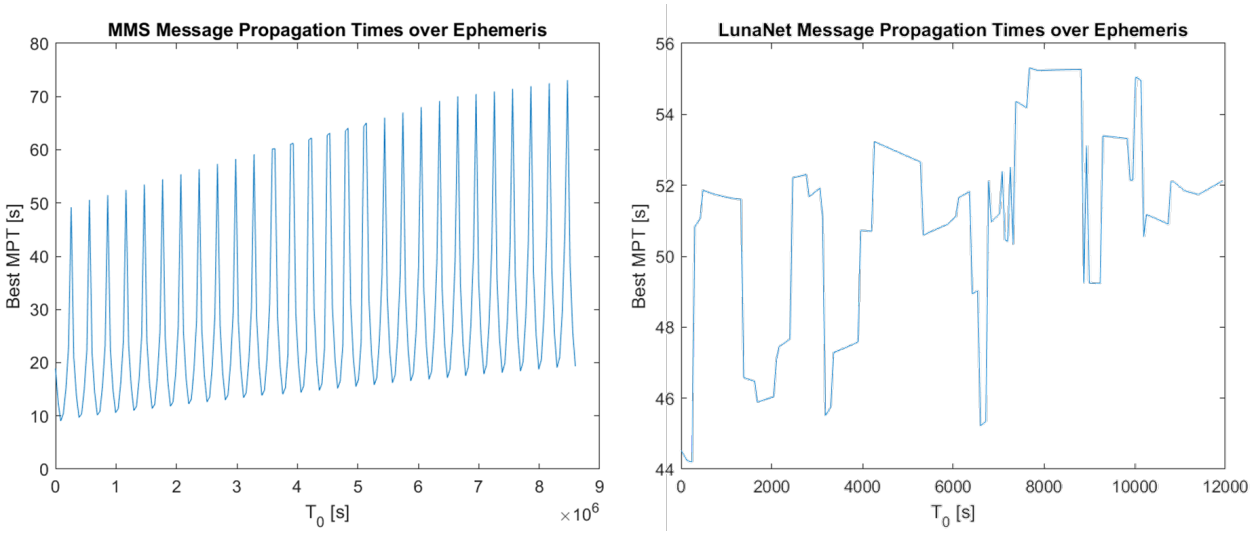

Fig. 11 The best MPT output by the routing-scheduling algorithm over time for the (left) MPT and (right) LunaNet constellations.

1. On the slant range-data rate relation.

Note that the results shown here use message propagation times based on the data rates given by the to the slant range-data rate table in Table 1.

Table 1 Binned slant range (km)-data rate (kbps) relation.

	Slant Range (km)	Data Rate (kbps)
i	0 < x </=5	50
ii	5 < x </=300	10
iii	300 < x </= 6000	1
iv	x > 6000	0.125

However, we also ran the algorithm using an exponential fit of the data in the table in order to generate a continuous mapping of slant range to data rate. Specifically, we run a least squares regression on the logarithm of the data rates and slant ranges, to obtain the approximate constants $a \approx 100.08kb \cdot [km^{0.667} \cdot s^{-1}]$, $b \approx 0.667$ in the relation

$$\text{data rate}[kbps] = \frac{a}{(\text{slant range}[km])^b}.$$

We find that in the development and comparison of message routing algorithms, using the exponential fit to determine message propagation times across edges allows for a more varied and a richer space to optimize over and compare outputs. We empirically observe that the more varied range of values from the exponential fit also yields better simulated annealing results.

IV. Near-Hamiltonian Cyclic Routing

As an alternative approach to optimization in the particular case of many-to-many communication scenarios, we explore a technique in which we concatenate a chain of messages around a **cycle**, until eventually every node in the network has received a complete set of messages from every other node. The key idea of this approach is that, rather than routing messages along a shortest path tree, we concatenate a chain of messages along a Hamiltonian cycle. This eliminates the need to solve both a routing problem and a scheduling problem: once we have identified a cycle, our scheduling procedure is to simply fire every other link at each timestep (described in more detail below). A *Hamiltonian cycle* is a path, starting and ending at the same satellite, which visits every satellite in the constellation exactly once. Within our constellations of interest, it is often the case that although the satellites and their lines of sight do form a connected graph, no Hamiltonian cycle exists. (In our simulated LunaNet constellation, for instance, sampling every 60 seconds over the course of a day, we found that a Hamiltonian cycle existed approximately 77 percent of the time). Instead of routing along a Hamiltonian cycle, then, we can route along a *near-Hamiltonian* cycle, in which the path still visits every satellite, but may visit some satellites more than once.

An additional consideration in this method is that the problem of determining whether a graph contains a Hamiltonian cycle is NP-hard. To search by brute force for a Hamiltonian cycle, the number of possibilites we must explore will grow factorially. This quickly becomes impractical when our satellite constellation contains more than a handful of nodes. Thus, we employ a brute-force technique to find the optimal solution as long as the total number of nodes is small (less than or equal to 4, a figure chosen heuristically based on observed performance), but switch to a greedy algorithmic approach when the number of nodes is larger.

A. Cycle-Finding Algorithm: Brute Force Version

Algorithm 3 CycleFinder

Input: A graph of the satellites and their current co-visibilities
Output: An efficient Hamiltonian or near-Hamiltonian cycle, computed via brute force

1: Generate all candidate cycles of length at most 2*nodes via permutations of 2*nodes elements, enforcing an upper bound of 2 on number of revisits to a single node
2: Check feasibility of each candidate cycle, based on visibility graph
3: For each feasible candidate cycle, compute time-cost of many-to-many message-sending via firing of alternating links
4: Return lowest-cost candidate cycle

B. Cycle-Finding Algorithm: Greedy Version

Algorithm 4 CycleFinder

Input: A graph of the satellites and their current co-visibilities
Output: An efficient Hamiltonian or near-Hamiltonian cycle, computed greedily

1: Start cycle at random node
2: **while** cycle not found **do**
3: From current node, compute 0.5*linkcost/avg + 0.5*neighbors/avg heuristc for each neighbor
4: Add lowest-cost neighbor to cycle as next-to-visit; mark as current (only consider nodes which have not yet been revisited, i.e. enforcing a "revisit cap" of one)
5: If we end up in a "dead-end", backtrack and choose second-lowest-cost neighbor instead
6: If cycle length is greater than 4/5 times number of satellites, raise revisit cap by one for all nodes
7: **end while**
8: Return found cycle

C. Scheduling Algorithm

A major motivation for considering this approach to the crosslink communication problem was that, once we have found a suitable cycle, no further computation time must be dedicated to determining a routing protocol. We deploy a pre-determined scheduling procedure, with slight variations based on two criteria: i) is our cycle perfectly Hamiltonian (i.e. has no revisits), and ii) does our cycle have even or odd length (measured in number of links)? We will describe this scheduling algorithm via examples illustrating the possible cases.

In the simplest possible case, wherein i) there are no revisits and ii) the found cycle has even length, our scheduling procedure is to simply fire every other link at each timestep, as demonstrated in the following example. Note that there is a "bottleneck" consisting of the maximum of the quantity *message size divided by transmission rate*, taken over all nodes firing in a given timestep. The length of the timestep is determined by this bottleneck.

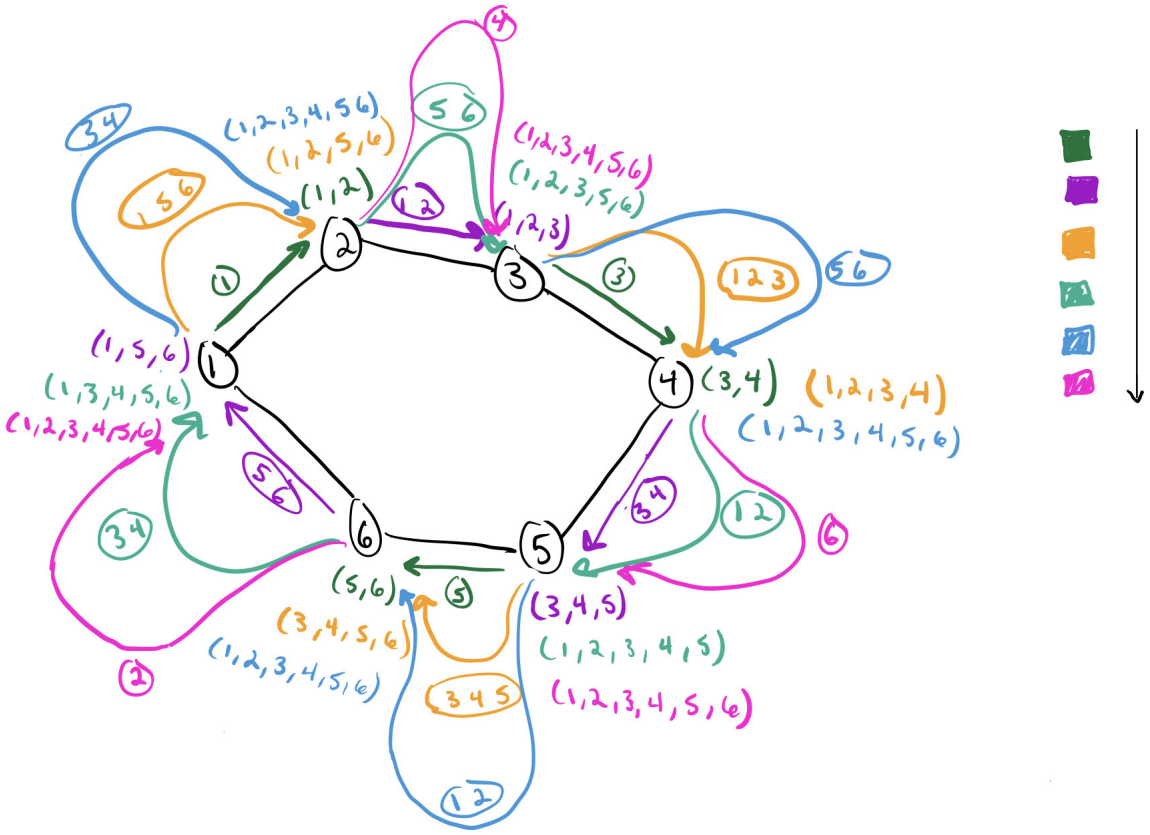

Fig. 12 Scenario 1: Message propogation in an even-length cycle with zero revisits. Information flows in timesteps with order indicated by the arrow. Circled numbers indicate passed messages at the timestep corresponding to the color used; numbers in parentheses indicate information possessed by a node at the timestep corresponding to the color used.

If either of the conditions i) no revisits or ii) even cycle length does not hold, the situation becomes slightly more complicated, but still there is no intensive computation which must be carried out to determine the message-sending schedule. In the case where the first condition (no revisits) holds, but the second (even cycle length) does not, we employ the following variation of the scheduling algorithm. Rather than dividing the links into two groups and firing all nodes in one of groups at each timestep, we divide the links into **three** groups and fire all nodes in one group at each timestep: we choose the lowest-cost node to fire in its own timestep, and then divide the rest of the nodes into two groups as before. The motivation here is to minimize the waiting time which the other nodes must perform as this lone link fires.

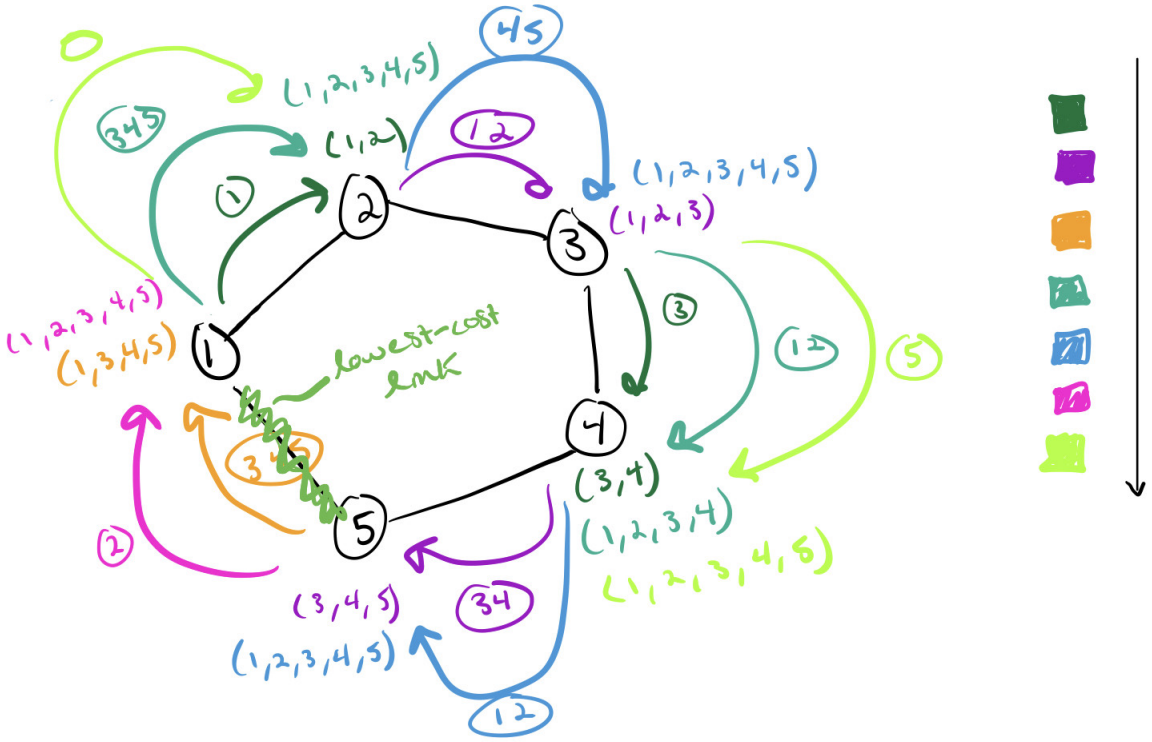

Fig. 13 Scenario 2: Message propogation in an odd-length cycle with zero revisits. Information flows in timesteps with order indicated by the arrow. Circled numbers indicate passed messages at the timestep corresponding to the color used; numbers in parentheses indicate information possessed by a node at the timestep corresponding to the color used. Notice that the lowest-cost link always fires within its own timestep.

In the case where the second condition (even cycle length) does hold, but the first (no revisits) does not, we employ a different slight variation of our original scheduling procedure. Here, we are restricted even further by the condition that each satellite may only perform a single action (sending/transmitting any message) at a given time; splitting the links into two groups is no longer enough to avoid violating this condition. Now, after splitting our links into two groups based on the parity of their positions as before, we employ a queuing system within these two groups. At a given timestep, we select greedily (by choosing those first reached when walking along our cycle) a set of links to fire first within the first, then greedily select a subsequent set of links, and so on until all of the links in the first group have fired. We repeat this procedure for the second group. In the example below, the cycle is effectively divided into **four** groups of links instead of the previous two by this procedure.

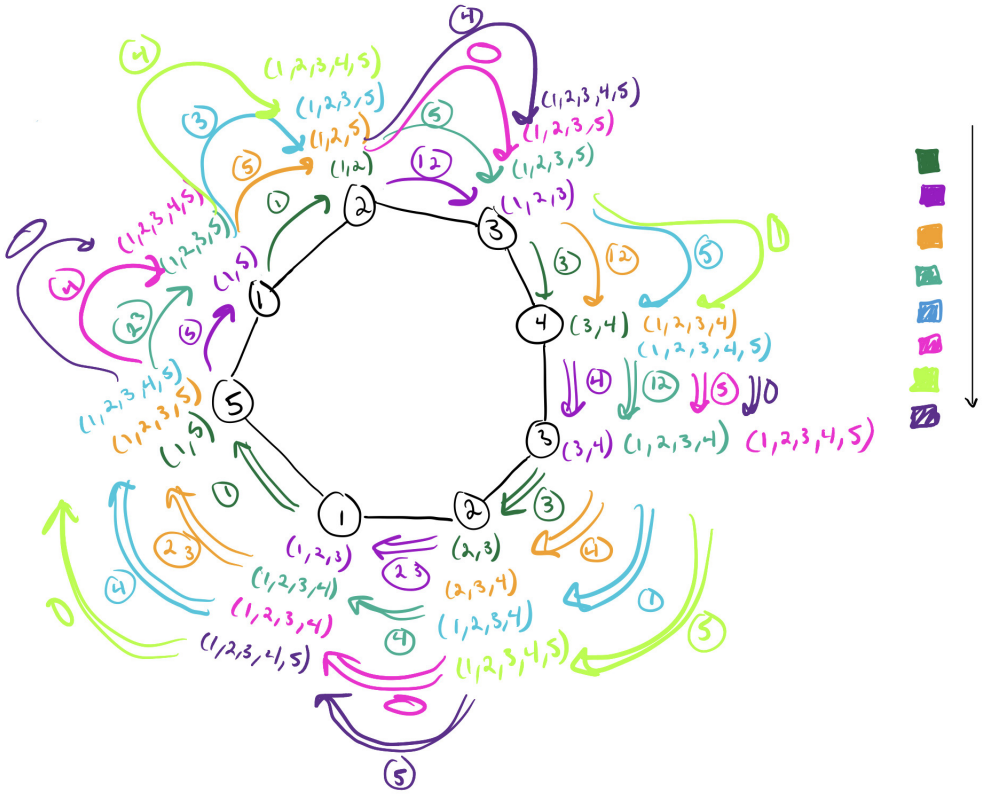

Fig. 14 Scenario 3: Message propogation in an even-length cycle with revisits. Information flows in timesteps with order indicated by the arrow. Circled numbers indicate passed messages at the timestep corresponding to the color used; numbers in parentheses indicate information possessed by a node at the timestep corresponding to the color used. Double-lined arrows indicate events which begin after all other events in a given timestep.

D. Results

We present a summary of this second algorithm's output in each of the four satellite configurations. On the MMS constellation, the brute-force version of the algorithm was run, as there were only four nodes to consider. In all other cases the greedy algorithm was run. It is notable that, while there is no variation at all in cycle length (and correspondingly number of revisits) within most constellations, inspection of the generated cycles themselves reveals that there is considerable variation in the actual nodes which are visited for all constellations except MMS (see GitHub, and in particular the script *cyclefinder backtrack heuristic.m*, for the code to generate cycles at each timestep). Lack of variability in cycle length and/or topological character may be partially accounted for by the fact that the algorithm was run on snapshots from the same orbits, with the same initial data, with only a small separation in time (i.e. on the order of minutes/hours/days).

Constellation	Number of Nodes	Average Cycle Length (links); stdev	Average Cost (seconds); stdev
MMS	4	4; 0	62.67; 34.5
LunaNet	5	6.17; 0.66	156.46; 18.26
ATrainCTrain	6	11; 0	127.3779; 2.65
GPS Walker	24	25; 0	1093.0; 194.2

We additionally present a graph of the MMS time-cost over approximately 200 timesteps, noting that the cost on average is higher than that attained by the alternating minimzation algorithm:

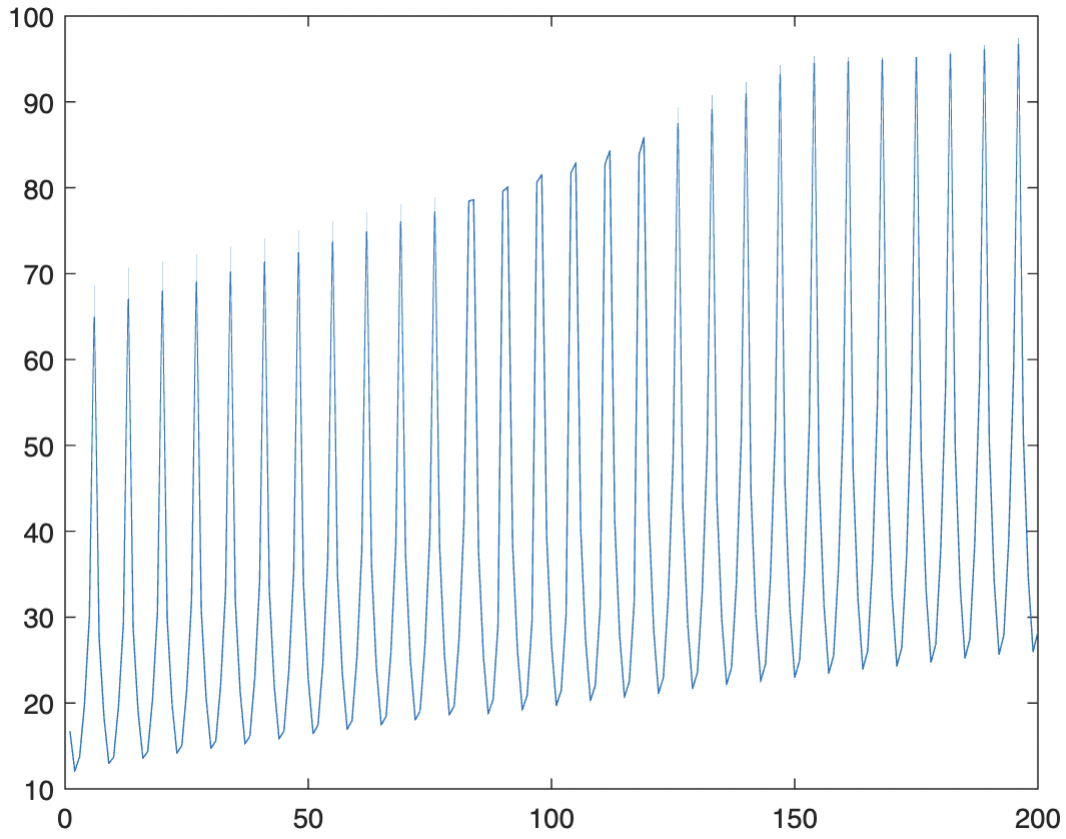

Fig. 15 MMS many-to-many transmission time (in seconds) vs. timestep, as computed by the brute-force cyclic algorithm.

E. Optimizations and Future Work

There are a number of interesting optimizations we may make to our cycle-finding algorithm, with the aim of optimizing performance under our pre-defined cyclic scheduling procedure. Once again, note that rather than optimizing routing and scheduling in tandem as in the previous approach to the problem, we are taking the approach of sovling the routing problem and then applying a fixed scheduling algorithm to the route we have found. As a result, we focus our optimization efforts on the cycle-finding stage.

–forcing an odd cycle. sometimes we might be able to achieve this by removing a node (test how often this is the case!). but otherwise, how does this trade off with adding in another revisit? –by limiting number of revisits, what bounds can we place on number of "splits" within one group? –pruning –discussion of bottlenecks. routing around? –small clusters –genetic tuning of heuristic –fine-tuning the time division procedure in the case where revisits occur (is there a better way to choose divisions than greedily?)

V. Discussion

A. Comparison of two approaches

- Run time: The near-Hamiltonian cycle approach to routing takes considerably less time to run than the alternating minimization algorithm for large ($\gtrsim 10$) nodes. Indeed, the alternating minimization algorithm is prohibitively slow on the 24-node GPS-Walker constellation. However, for smaller graphs and for simpler message passing settings, it is worth running both approaches and comparing results.

Recommendations for the different message passing settings:

- One-to-one: standard shortest-path routing
- One-to-many, many-to-one: alternating minimization As number of nodes grows, can remain efficient by switching from alternating minimization to shortest-path tree with first-in-first-out scheduling
- Many-to-many: greedy cyclic routing for large networks; alternating minimization and/or greedy cyclic for small networks In all other scenarios, enforcing visitation to every node for each message is excessive/unnecessary. But in many-to-many, cyclic structure enables efficient network-sharing, as long as revisitations are minimal (heuristic should prioritize this!)

B. Takeaways

- Use and choice of heuristics for optimization problems: Heuristics are important in developing greedy algorithms to get approximate solutions to NP-Hard problems, as well as estimating the value of a parameter before it can be fully calculated.
- Reducing and decomposing problems to known problems with efficient approximation algorithms (e.g., reducing scheduling to Max Independent Set Problem)
- Decomposing a hard multi-part problem into more manageable pieces (e.g. alternating minimization problem)
- Tradeoff between optimal solutions which require high computing power (e.g. brute force search), vs. "good enough" solutions which do not (e.g. greedy suboptimal algorithms)

C. Future work and considerations

Thus far, we have only worked with a single connected graph component, and assumed that the constellation graph is constant over the course of a message passing session. In future work, we would like to consider the setting of changing connectivity of the graph, specifically, different having connected components over time. We may then need to coordinate the transmission of a message not only over nodes in a given configuration, but over nodes in time.

Appendix

All code used is available on our GitHub repository, `https://github.com/daviddezellturner/crosslink_topologies`.

D. Graph profiles over time

Figures 16 - 18 show how the access profiles and slant ranges of the LunaNet, A-Train C-Train, and MMS constellations change over time. Our message routing algorithms assume that the weighted adjacency matrix of the graph is approximately fixed over the course of message passing, which takes no more than a couple of minutes.

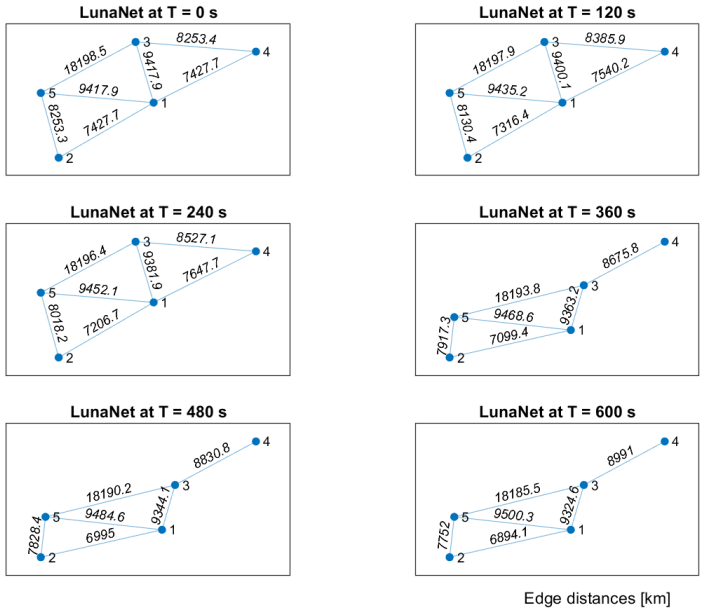

Fig. 16 LunaNet constellation graphs with edge weights representing the slant range in km between given nodes with access, at various points of the ephemeris.

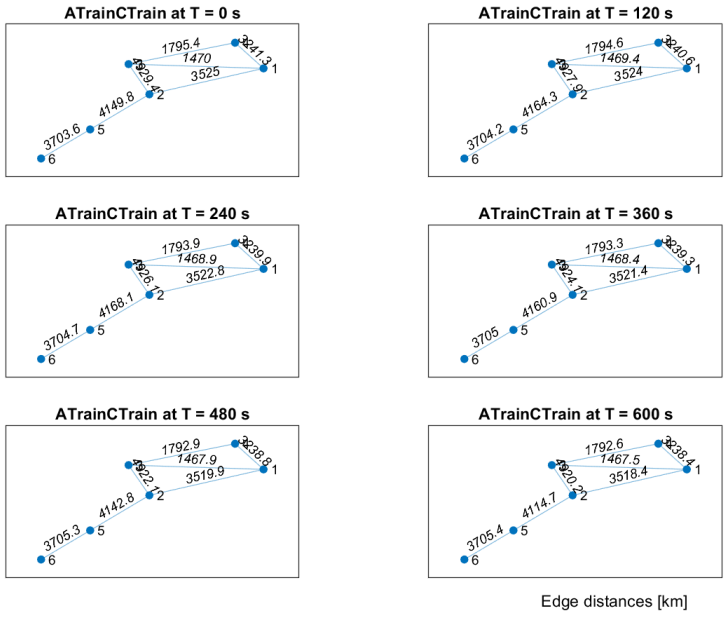

Fig. 17 A-Train C-Train constellation graphs with edge weights representing the slant range in km between given nodes with access, at various points of the ephemeris.

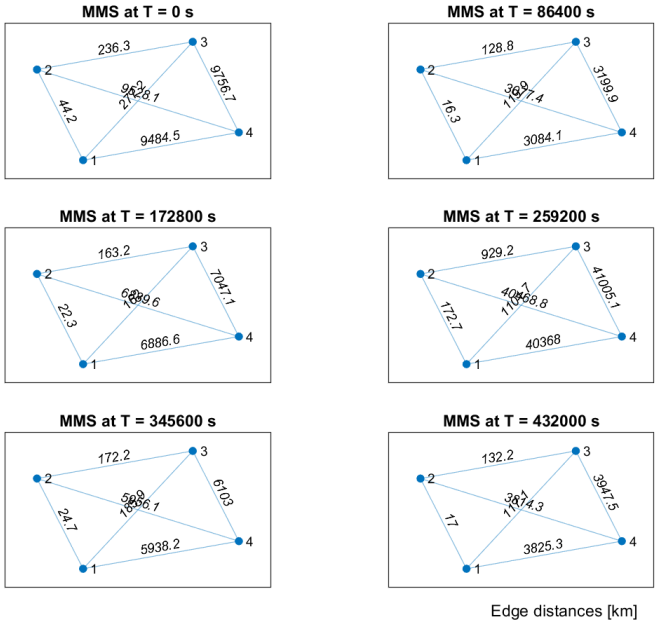

Fig. 18 MMS constellation graphs with edge weights representing the slant range in km between given nodes with access, at various points of the ephemeris.

References

[1] National Aeronautics and Space Administration, "LunaNet Concept of Operations and Architecture," 2020.

[2] Cheung, K.-M., and Lee, C., "Lunar relay coverage analysis for RF and optical links," *2018 SpaceOps Conference*, 2018. https://doi.org/10.2514/6.2018-2612.

[3] Mazarico, E., Rowlands, D. D., Neumann, G. A., Smith, D. E., Torrence, M. H., Lemoine, F. G., and Zuber, M. T., "Orbit determination of the Lunar Reconnaissance Orbiter," *Journal of Geodesy*, Vol. 86, No. 3, 2011, p. 193–207. https://doi.org/10.1007/s00190-011-0509-4.

[4] Sakai, S., Togasaki, M., and Yamazaki, K., "A note on greedy algorithms for the maximum weighted independent set problem," *Discrete applied mathematics*, Vol. 126, No. 2-3, 2003, pp. 313–322.