Routing in Scalable Delay-Tolerant Space Networks with Graph Neural Networks

Matías Olmedo, Juan A Fraire, Renato Cherini, Fabio Patrone, Mario Marchese

Routing in Scalable Delay-Tolerant Space Networks with Graph Neural Networks

Matías Olmedo
Instituto Gulich, CONAE,
Córdoba, Argentina
mati.olmedo95@gmail.com

Juan A. Fraire
Univ Lyon, Inria, INSA Lyon, CITI,
69621 Villeurbanne, France
CONICET, Córdoba, Argentina
juan.fraire@inria.fr

Renato Cherini
Universidad Nacional de Córdoba,
Córdoba, Argentina
renato.cherini@unc.edu.ar

Fabio Patrone
University of Genoa, Genoa, Italy
f.patrone@edu.unige.it

Mario Marchese
University of Genoa, Genoa, Italy
mario.marchese@unige.it

Abstract

Space networks are challenged by long propagation delays and prolonged link disruptions, which require Delay-Tolerant Networking (DTN) mechanisms. Contact Graph Routing (CGR) exploits the a priori available topology knowledge derived from the predictable trajectories to compute optimal end-to-end routes. However, state-of-the-art CGR is limited in the scalability and stability of the computation effort, which is critical in resource-constrained spacecraft. To overcome this issue, this paper presents GAUSS: a graph neural network-based routing for scalable delay-tolerant space networks. By harvesting recent advances in Graph Neural Networks (GNNs), we can improve the scalability of CGR by a factor of two while narrowing the variability down to one-third in realistic cislunar and near-Earth systems.

1 Introduction

The space sector is thriving. Recent advances in miniaturization and distributed systems, combined with reduced costs, allow new actors and missions to be deployed in space [29]. Networked missions are becoming more popular in near-Earth and deep-space contexts where prolonged disruptions and long propagation delays characterize links.

For these kinds of networks, Delay-Tolerant Networking (DTN) [30] offers appealing store-carry-and-forward mechanisms, including routing and forwarding schemes such as Contact Graph Routing (CGR) [3]. CGR leverages the predictable nature of space assets, for which expected trajectories and visibility can be computed in advance. Thus, future *contacts* between participating spacecraft can be provisioned and imprinted in a *contact plan* comprising information on the forthcoming connectivity. Based on the contact plan, routes with optimal delivery time can be computed on-demand on each node [15].

However, the state-of-the-art on CGR routing is limited in compute time *scalability* and *stability*. On the one hand, related research has shown that CGR scales beyond practical use with the number of nodes and contacts in the contact plan [31]. This forbids the conception of a large-scale fleet of nodes equipped with constrained onboard computers (lack of air impedes heat dissipation, which forces running under-clocked processors). On the other hand, the computing effort to determine routes toward different destinations highly depends on the topology and when the route is demanded. Therefore, route computation time unpredictably deviates from the average during the mission, complicating spacecraft resources' overall scheduling and operation.

To overcome these CGR issues, the present work investigates supervised Machine Learning (ML) approaches as a solution to capture the routing intelligence in a model trained on the ground. The underlying hypothesis is that route determination time on board will be shielded from the contact plan size, thus delivering a close-to-optimal path in bounded and stable time frames. To achieve this goal, we leverage Graph Neural Networks (GNNs), which perfectly fit CGR data models based on graph-structured data with sparse contact elements and uneven relationships. Our solution is coined GAUSS: graph neural networks-based routing for scalable delay-tolerant space networks [1] While previous works have approached routing in space DTNs with diverse ML techniques [8–10, 12, 20–23, 27, 32], to the best of our knowledge, this is the first to focus on the scalability and stability of the on-board routing computation effort. We evaluate GAUSS' scalability and stability in realistic near-Earth and cislunar DTN topologies.

The remainder of this paper is organized as follows. Section 2 revises related works in the area of DTN, ML, and GNNs. The GAUSS model is presented in Section 3. Section 4 discusses evaluation outcomes from realistic scenarios. Conclusions are summarized in Section 5.

[1] Code available in `https://github.com/matiolmedo/GAUSS-Code`

2 Background

2.1 Space DTNs

The DTN term was first introduced in [3] to identify communication networks that cannot provide continuous (e.g., persistent) end-to-end communications. In other words, Internet-like multi-hop paths characterized by low and stable latency, low error rates, and high/symmetrical bandwidth cannot be assumed. In addition to satellite communication networks, DTN has many other applications, such as the Internet of Things, Vehicular Ad-Hoc Networks, Submarine Networks, and Mobile Social Networks.

The core approach in DTN is to allow data to be temporarily stored in relay nodes and forward it in a *store-carry-and-forward* fashion exploiting different degrees of contact predictability. In the DTN architecture specification [30], a *contact* is defined as a period in which two nodes that belong to the network can exchange data between them. In turn, contacts are classified as (*i*) *opportunistic*: no assumptions can be made about future contacts, (*ii*) *probabilistic*: contact patterns can be inferred from history (e.g., social media), and (*iii*) *scheduled*: contacts can be accurately predicted and documented in a contact plan.

Space and satellite network systems are governed by predictable orbital mechanics, which makes them a perfect fit for DTNs with scheduled contacts. Space networks are classified into a) deep space networks and b) near-Earth networks. Deep space networks span rovers, orbiters, and probes in beyond-Lunar space, where the signal propagation delay is the dominant effect, combined with link outages provoked by planetary occlusion. Networks in near-Earth environments comprise satellites in low, medium, and geostationary Earth orbits (LEO, MEO, and GEO satellites, respectively) where link disruptions due to range/line-of-sight constraints are frequent. However, the propagation delay is negligible in near-Earth systems, as discussed in [5–7]. Note that this paper focuses on sparse near-Earth networks (reduced amount of spacecraft with significant connectivity gaps) and not mega-constellations (hundreds or thousands of satellites with persistent end-to-end paths). The reader is referred to [17] for a sharper definition of sparse and dense near-Earth constellations.

2.2 DTN Protocols and Routing

The DTN architecture can be implemented through the Bundle Protocol [26]. The Bundle Protocol adds a "Bundle layer" above the transport layer and below the application layer. The layer offers persistent storage of *bundles* (the DTN protocol data unit) to overcome network outages. Devices implementing the Bundle protocol are called DTN Nodes. Routing in DTN implies the determination of routes with a specific time dimension, i.e., valid within a specific time interval. Flooding-based approaches, such as Epidemic and Spray-and-wait [28], exploit replication over opportunistic contacts. Probabilistic contact approaches, such as Prophet [24], incorporate abstract topology models to reduce replication and improve delivery metrics. However, the computation of more precise time-dynamic aspects is facilitated in scheduled DTNs using the so-called *contact plan*, which includes the forthcoming network connectivity [16].

Graph-based algorithms, such as Contact Graph Routing (CGR), stand out since they have been able to demonstrate precision (no replication) and efficiency (lowest delivery time) in the routing task for space DTNs. CGR was introduced in [3]. Later work proposed using Dijkstra and Yen's algorithm to compute route tables with the K-best routes [18], the approach currently supported by the Interplanetary Overlay Network (ION) DTN stack developed by NASA [2]. The core challenge of the current CGR is its scalability. As discussed in [31], CGR computational complexity grows significantly with the number of contacts in the contact plan, where large-scale networks are practically unfeasible. On top of this, routes are only valid for a given period on which the next-hop node is the best neighbor to forward a bundle to a given destination. Also, route data volume can be booked by local or remote traffic, which demands the utilization of alternative paths [15]. This means multiple routes must be computed frequently on resource-constrained onboard computers.

2.3 Routing on DTN with ML

Several algorithms for routing in DTNs based on different ML techniques and learning strategies have been proposed.

Q-Routing [10, 13, 22] and *Q-Learning* [21, 32] are the most prominent approaches using *reinforcement learning*. In this context, the model interacts with a scenario to discover how to maximize specific metrics, typically the end-to-end packet delivery time. *Deep Q-Learning* [4,9,12], a Q-Learning extension based on deep neural networks, increase the generalization capability of this kind of model. Although not explicit in all cases, Q-routing models assume a learning phase. Learning can occur on-ground using simulated environments before deploying the resulting parameters to the space network.

There exist multiple DTN routing approaches using ML techniques in the context of *supervised learning*, as Decision Trees [8, 11, 25, 27] and Neural Networks [20, 23]. These works consider routing a classification problem, aiming to predict the best neighbor node to be used as the next hop according to specific metrics (e.g., buffer occupation, successful deliveries, node speed/power, distances, current hop count, message lifetime, etc.). Also, supervised ML models are trained on the ground and then provisioned to the spacecraft. In general, mentioned ML-based routing approaches did not exploit the graph-based structure of the contact plan and were not designed to improve the scalability and stability of the onboard route computation processes.

2.4 Graph Neural Networks

Graph Neural Networks (GNN) are powerful machine learning models with a strong inductive bias [1] that allow the processing of graph-structured data. GNNs are best suitable when the problem domain includes sparse elements and uneven relationships between them. In contrast to other models devoted to linear or homogeneous structured data, like Multi-Layer Perceptrons (MLP), Recurrent Neural Networks (RNN), or Convolutional Neural Networks (CNN), GNNs explicitly use the heterogeneous structure of data during learning. Over the years, different approaches have been presented, such as Graph Convolutional Networks (GCN),

Graph Attention Networks (GAT), and Graph Recurrent Networks (GRN), among others. They have proven to solve tasks effectively, including rich relational structure in various domains, and to use different learning strategies. We refer to [1] for a comprehensive review of models and applications.

In 2017, Gilmer *et al.* introduced a general framework called Message-Passing Neural Network (MPNN) [19], which unifies most of the previous approaches into a standard setting. In the most basic operation of a GNN, an initial state is associated with every graph element (nodes and edges) using input information. Then, each element state is iteratively updated, combining the previous state and the previous state of related elements according to the graph structure. In the end, the final states of graph elements are used by a domain-specific function to produce the output for the given task. MPNNs use an iterative message-passing algorithm to propagate information between nodes. In each message-passing step t, a node v receives messages from every node w in its neighborhood ($N(v)$). Messages are constructed using the paired nodes states h_v^t and h_w^t, the state of the edge between them e_{vw}^t, and a shared function M^t. These parameters are thus combined by using an invariant to permutations aggregation function $\bigodot$: $m_v^{t+1} = \bigodot_{w \in N(v)} M^t(h_v^t, h_w^t, e_{vw}^t)$. The state of v is updated by using the previous state h_v^t, aggregated messages m_v^{t+1} and a shared update function U^t: $h_v^{t+1} = U^t(h_v^t, m_v^{t+1})$ where M^t and U^t, which are specific for each iteration t, are learned differentiable functions, typically MLPs. The fact that all the nodes in a graph share these functions is the key factor that enables us to keep the model size tractable. It also allows using the same model to process graphs with different structures and numbers of nodes. After a certain number of iterations, the output (ranging from a single value to a graph with node and edge attributes) is computed using a so-called *readout* function, usually an MLP, predicting the given task.

Later, in the aforementioned work [1], Battaglia *et al.* presents a comprehensive study about inductive biases on deep learning and introduces a broader framework called Graph Networks (GN) *"for relational reasoning over graph-structured representations"*. In this setting, the primary component is the Graph Network block, which takes a graph as input and returns an updated version. Typically, GN blocks are stacked sequentially, resembling the iterative processing of MPNNs. However, the GN framework is more general than the MPNN since it provides mechanisms for updating nodes, edge states, and a global state. In its more general form, a GN block comprises three *update functions* ϕ^e, ϕ^v, ϕ^u for edges, nodes, and global state, respectively, given by the following equations:

$$\begin{aligned} \mathbf{e}'_k &= \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}) & & \bar{\mathbf{e}}'_i = \rho^{e\to v}(E'_i) \\ \mathbf{v}'_i &= \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}) & \text{where} \quad & \bar{\mathbf{e}}' = \rho^{e\to u}(E') \\ \mathbf{u}' &= \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}) & & \bar{\mathbf{v}}' = \rho^{v\to u}(V') \end{aligned} \tag{1}$$

Block processing starts updating edge attributes. For a given edge k from *source* node s_k to *receiving* node r_k, its updated attributes $\mathbf{e}'_k$ are computed using update function ϕ^e taking as inputs its previous attributes $\mathbf{e}_k$, receiving node attributes $\mathbf{v}_{r_k}$, source node attributes $\mathbf{v}_{s_k}$ and global state attributes $\mathbf{u}$.

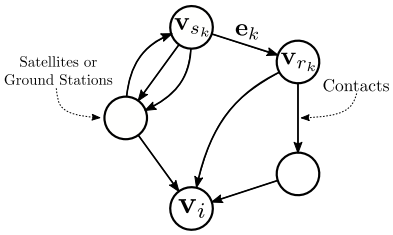

Figure 1: GAUSS graph

Node updating depends on aggregating edges attributes updated in the previous step. For a given node i, $\bar{\mathbf{e}}'_i$ is computed by an *aggregation function* $\rho^{e\to v}$ applied to the set of attributes of incident edges having i as receiving node, denoted by E'_i. The updated attributes for i, namely $\mathbf{v}'_i$, are computed using the update function ϕ^v taking as inputs $\bar{\mathbf{e}}'_i$, its previous attributes $\mathbf{v}_i$ and the global state attributes $\mathbf{u}$.

The updating of the global state depends on the aggregating edge and node attributes, both updated in the previous steps. The update function ϕ^u computes the updated global state attributes $\mathbf{u}'$ by using, as input, the aggregated attributes of every edge $\bar{\mathbf{e}}'$, the aggregated attributes of every node $\bar{\mathbf{v}}'$ and the previous global state attributes $\mathbf{u}$.

A general GN block is depicted in Figure 2 (a), where V denotes the set of nodes (attributes), E the set of edges (attributes), $\mathbf{u}$ the global state (attributes) and V', E', $\mathbf{u}'$ the updated versions of them. A GN block can be configured to achieve different goals. Excluding one or more updating functions or even discarding some function inputs allow the block to focus on specific elements of the graph, as depicted in Figure 2 (b) and (c). Combining differently configured blocks gives rise to diverse architectures to process graphs and predict local properties of edges and nodes or even global properties of the complete graph. This flexibility makes GN the most general framework for constructing neural network models over graph-structured data.

In the following section, we propose a GN-based model to improve the computational tractability of CGR.

3 GAUSS

3.1 GAUSS Graph Model

In GAUSS, a graph represents a transmission demand from a given source node to a certain destination node in a delay-tolerant network composed of satellites and ground stations, with intermittent directed communication links between each other during a certain period. More precisely, a directed multigraph $G = \langle V, E \rangle$ with $V \in \mathbb{N} \to \mathbb{R}^{k2}$ an indexed set of attributed vertices representing DTN nodes (vertices and nodes are henceforth used interchangeably) and $E \in \mathbb{N} \to \mathbb{R}^l \times dom(V) \times dom(V)$ an indexed set of attributed edges representing time-bounded contacts between the DTN nodes (edges and contacts are henceforth used interchangeably). Multiple edges between a pair of vertices could ex-

[2] $\mathbb{N}$ denotes the set of natural numbers, $\mathbb{R}^n$ the n dimensional Cartesian product of real numbers, $\to$ the set of partial functions and $dom(f)$ the domain of function f.

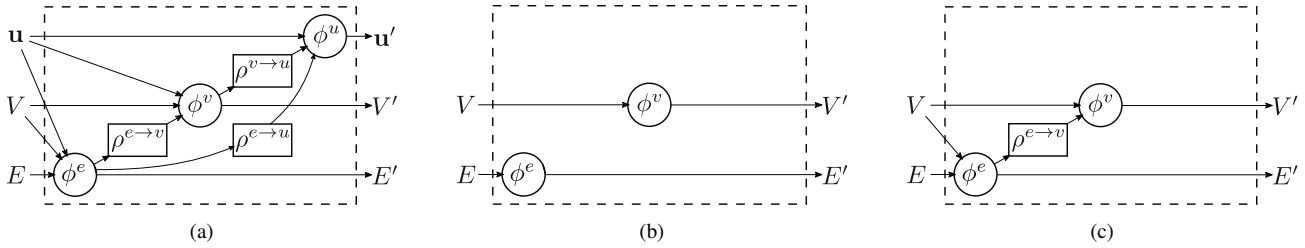

Figure 2: (a) Full block, (b) Encoder/decoder block, (c) Core block.

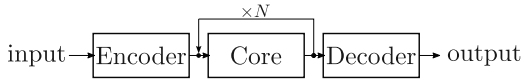

Figure 3: GAUSS Processing Pipeline

ist since two DTN nodes can experience several contacts throughout the considered period of time. Edges are directed since space communications are often limited and not necessarily bidirectional or symmetric.

Figure 1 illustrates a GAUSS graph. $\mathbf{v}_i$ denotes the attributes of the node with index i (i.e., $V(i) = \mathbf{v}_i$) and $\mathbf{e}_k$ the attributes of the edge with index k, whose sending and receiving node indexes are denoted by s_k and r_k, respectively (i.e., $E(k) = (\mathbf{e}_k, r_k, s_k)$). In the case of input graphs, attributes of vertices are 2-dimensional vectors containing a one-hot codification that indicates which node is the sender ($[0,1]$) and which is the receiver ($[1,0]$). Every other node is initialized with $[0,0]$. Edge attributes are also 2-dimensional vectors containing the start and end times of the contact ($[t_{start}, t_{end}]$) codified as time units since the beginning of the considered period, which is assumed to initiate with the start of the demand. Output graphs represent the path through contacts and nodes which route the demand. Vertex and edge attributes are 2-dimensional vectors containing a score of the chance the element is *member* of the solution path (first component) or not (second component). Thus, ground truth graphs annotate nodes and edges using one-hot encoding, where $[1,0]$ codifies that the element is part of the solution and $[0,1]$ otherwise.

As usual, inner graphs occurring in intermediate computation steps use higher dimensional vectors as attributes, providing the model with enough features to allow effective training. In the experiments, 16-dimensional vectors were sufficient for both edges and nodes.

3.2 GAUSS Architecture

GAUSS uses an *encode-process-decode* pipeline architecture, as illustrated in Figure 3. Recalling equations 1, GN block configuration is described below. It is worth noting that the global state is not used in this version of GAUSS.

Encoder block transforms edge and node attributes of the input graph into a higher dimensional latent representation without taking into account any relationship between elements, as illustrated in Figure 2 (b). This is accomplished by both 2-layer MLP according to the following equations: $\mathbf{e}'_k = \text{MLP}_e(\mathbf{e}_k)$, $\mathbf{v}'_i = \text{MLP}_v(\mathbf{v}_i)$.

Table 1: Near-Earth Scenario Parameters

Type of Orbits	Circular
Inclination	53°
Propagator	TwoBody
Maximum range constraint	2500 Km
Ground Station (CETT, Argentina)	Lat: -31.525, Lon: -64.462
Constrain of minimum elevation in GS	5°
Number of messages (Train, Test, Eval)	10240, 5120, 5000
Period for Train, Test, Eval [hrs]	0 - 96, 96 - 192, 96 - 192

Scenario	LEO A	LEO B	LEO C	LEO D
Orbital planes	2	2	4	3
Sats. per plane	12	16	16	24
Orbital height	400, 500 Km	400, 500 Km	400, 500 Km	400, 450 500 Km
Δ True anomaly	60	20	90	20
Constellation	Ring	Train	Ring	Train

Core block performs the main processing over a loop of N iterations (we consider $N = 5$), receiving as input a graph with element attributes resulting from concatenating the Encoder output and previous iteration output of the block itself. This reinforces the original input information in each processing step, improving the network's training. Update functions are implemented as both 2-layers MLP and edge aggregation function as a point-wise summation, according to the following equations: $\mathbf{e}'_k = \text{MLP}_e([\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}])$ and $\mathbf{v}'_i = \text{MLP}_v([\bar{\mathbf{r}}'_i, \bar{\mathbf{s}}'_i, \mathbf{v}_i])$. where $R'_i = \{(\mathbf{e}, r, s) \in E' \text{ s.t. } r = i\}$ is the set of edges to node i, $S'_i = \{(\mathbf{e}, r, s) \in E' \text{ s.t. } s = i\}$ is the set of edges from node i, E' is the updated set of edges, and $[\,]$ denotes vector concatenation. MLPs are shared across iterations.

Decoder block is configured analogously to Encoder block but using both MLP with an extra final layer to decrease the attributes dimensionality to produce the expected output.

3.3 GAUSS Dataset Synthesis

We synthesize datasets from four near-Earth scenarios and four realistic topologies of Lunar networks.

Near-Earth Scenarios Specifications for the Near-Earth case are summarized in Table 1. Each scenario consists of up to 40 satellites and 1 ground station and lasts for 192 hours. The orbital dynamic on the sparse constellation forces sporadic inter-satellite link (ISL) opportunities. As a result, data must flow in a store-carry-and-forward fashion from satellite to satellite and finally to the ground station. We derive con-

tact plans leveraging the Two Body orbital propagator to determine the time-evolving position of spacecraft, taking into account the initial orbital parameters, gravitational forces, atmospheric drag, and solar pressure (obtained from publicly available Two-Line Elements (TLE) files), and a 2,500 km slant range constraint for visibility condition.

For each near-Earth scenario, during the first 96 hours of its contact plan, we simulate 10,240 transmission demands which compose its *training dataset*. Each demand is generated considering a satellite as the source node and a start time that is randomly chosen from a uniform distribution. Without loss of generality, we assume downlink traffic. Therefore, the destination node is always the ground station. The demand is instantiated as an input graph considering the slice of the contact plan beginning at its start time and having a duration limited by a parameter t_{win}. This parameter limits and homogenizes the time window in which the model looks for a routing solution. The ground truth is generated by computing the best route obtained using the CGR algorithm. The demand is discarded if it is impossible to get a solution due to the time constraint. Using the same procedure described above, we consider the last 96 hours of the contact plan to simulate 5,000 demands which compose the *testing dataset* for the scenario. In this case, we limit the start time up to the contact plan end time minus t_{win} to give time to the demands to reach the destination.

Cislunar Scenarios

To study the application of GAUSS on deep space DTNs, we also consider the Lunar network topology presented in [14]. The goal of this scenario is for satellites in Lunar orbit to collect data from sensors deployed on the opposite side of the Moon. Since no direct-to-Earth communication is possible, data is stored temporarily in orbiting satellites until a link becomes available. However, since satellites are considered resource-constrained, dedicated Moon stations (on the visible side of the Moon's surface and equipped with solar panels) act as relay nodes. We define two scenarios (Moon A and Moon B, corresponding to scenarios S1 and S2 from [14]), each enabling a subset of 4 out of the 6 Moon ground stations. Furthermore, for four scenarios, we consider Moon A and B with and without ISLs between the Lunar orbiters. Please refer to [14] for the full scenario parameters. We obtain the contact plans from this scenario using a Lunar Two-Body propagator (Lunar orbits are deployed in stable inclinations [14]). The slant range for communications between Lunar orbiters and surface nodes is assumed 2000 km. Links from the Lunar station to Earth are considered persistent thanks to using NASA's Deep Space Network (DSN) antennas. For each of the four scenarios, we simulate 14 days of connectivity. The first 7 days are used as the *training dataset*, while the second 7 days are for the *testing dataset*. Traffic demands are generated from the Moon sensors towards the ground (any of the three DSN nodes), leveraging a uniform distribution for the generation time. The t_{win} parameter limiting traffic generation is 4.5 days.

3.4 GAUSS Training

We train a different GAUSS model for each scenario using its training dataset. The training dataset is labeled with the optimal CGR routing outcome, computed on the ground in advance. Thus, the GNN model shall capture CGR decisions, including its most advanced features, such as congestion mitigation and optimal route selection. The parameters of the trained model are then provisioned to the space nodes.

We are only interested in evaluating how a particular trained model generalizes with respect to routing demands along the predicted evolution of the scenario contact plan, in which only minor changes in the underlying topology are expected. Recall that input graphs contain only a slice of the contact plan; therefore, they typically differ in the occurrence of edges. But a model could even deal satisfactorily with changes in the occurrence of nodes. This flexibility is an intrinsic property of GNN models.

However, we do not consider it necessary to have a single trained model to account for scenarios with substantially different topologies. In the scenarios we are interested in, it is reasonable to train a model fitting a particular topology and traffic demand, providing better input performance according to the given topology context. In space networks, significant changes in an application scenario topology occur in specific situations and with relative foresight (e.g., spacecraft maneuver, satellite de-orbit, etc.). They could be dealt with by re-training the model and re-provisioning it in the routing components. From a concept of operations perspective, a GAUSS model can be updated periodically, just like a contact plan for a CGR-based system.

We train GAUSS under a supervised learning paradigm, aiming to learn the optimal weights of the block update functions that minimize the loss function defined as: $\mathcal{L} = \xi(\forall v \in V) + \xi(\forall e \in E)$ where ξ is the Softmax Cross Entropy function. The training process uses the Adam stochastic gradient descent and back-propagation and is terminated after 62 epochs. The batch size is set to 32 and the learning rate to 0.001. Non-linearity is introduced in every MLP by using Rectified Linear Unit as the activation function.

The training processes of the different GAUSS models took different computational efforts depending on the scenario used. The hardware used to obtain these values was an Intel i7 processor with 32 GB of RAM running an Ubuntu 20.04.2 LTS OS. For scenarios in Low Earth Orbit (LEO), the training times range from 8 hours for LEO A to 25 hours for LEO D. Moving on to scenarios involving the Moon, the training times increase significantly. For example, Moon A requires 18 hours of training, while Moon A ISL (Integrated Surface and Lunar) takes 24 hours. The training time further escalates for Moon B, with a duration of 49 hours, and Moon B ISL demanding the longest training time of 72 hours.

3.5 GAUSS Post-Processing

For a transmission demand at a source node, we use the output graph obtained from GAUSS to obtain the route to the destination node by using the following approach with almost constant computing cost. This is relevant as GAUSS route determination occurs onboard the satellites. Starting from the source node, the route is computed iteratively, choosing the contact with the higher *membership* score until reaching the destination. Algorithm 1 is applied to the output graph to obtain the best route $[R]$ for a given source-destination (v_{src}, v_{dst}) pair. The algorithm explores the graph

starting from v_{src} where the highest probability outgoing edge e_p is chosen. The next hop at the other end of e_p is now the current vertex v_{curr}, and the edge is added to $[R]$. The algorithm finishes with the complete end-to-end route encoded in $[R]$.

Algorithm 1: Best Route Determination

```
input: v_src, v_dst, Output graph
begin
    [R] ← []
    v_curr ← v_src
    while CurrentNode ≠ v_dst do
        e_p ← outgoing v_curr edge with highest prob.
        v_curr ← e_p.dst
        [R].append(e_p)
    end
    return: [R]
end
```

4 Evaluation

To evaluate the resulting algorithm performance, we study its *Efficiency*: the resources consumed to execute the routing routine (metric is the onboard computation time) and its *Efficacy*: the network performance metrics achieved by the routing scheme (metrics are packet arrival time and delivery ratio). In terms of these metrics, we compare the following routing approaches: (*i*) *GAUSS*: the proposed GNN-based routing scheme; (*ii*) *CGR*: the network performance baseline providing optimum delivery times, and (*iii*) *Direct*: the resource utilization baseline that forwards the demand from source to destination nodes by using the first direct contact between them (demands nearly zero computing effort). Note that we do not measure direct forwarding metrics in the Cislunar case since Moon nodes are on the opposite side of the Moon. In this case, no direct link is feasible. Thus, delivery time is infinite, and the delivery ratio is zero. It is worth mentioning that both GAUSS and *Direct* can fail to find a path to the destination, even when one is available. This will become evident in the delivery ratio metric. However, we design the LEO scenario contact plans to ensure direct contact between any pair of nodes, minimizing the possibility that *Direct* fails. As mentioned, this is not possible in the Cislunar scenario. This feature and the cyclical nature of scenarios lead to no significant difference in the number of unsatisfied demands using each method.

Table 2: Efficacy results. Arrival times for near-Earth and cislunar scenarios

LEO Scenario		A	B	C	D
CGR	Max [min]	709.9	718.9	527.9	484.2
	Mean [min]	200.2	136.0	121.0	82.5
GAUSS	Max [min]	719.9	719.9	718.8	719.8
	Mean [min]	278.6	248.8	232.8	265.3
Direct	Max [min]	905.3	889.8	890.4	897.4
	Mean [min]	291.3	289.3	298.3	294.0

Moon Scenario		A	A ISL	B	B ISL
CGR	Max [min]	1.8	1.8	1.8	1.8
	Mean [min]	0.3	0.3	0.3	0.3
GAUSS	Max [min]	1.9	1.9	1.9	1.9
	Mean [min]	0.8	0.3	0.4	1.4

4.1 Analysis

We evaluate the three approaches mentioned over each near-Earth and Cislunar scenario's *testing dataset*. For the efficiency results, we summarize the processing effort in Fig. 4, presenting histograms of the distribution of transmission demands concerning the time required to compute their routes onboard. Regarding efficacy results, the maximum and average packet arrival times for the Near-Earth and Cislunar scenarios are provided in Table 2. Fig. 5 blends the two former analyses by presenting the overall arrival time and delivery ratio. The average onboard processing time is also plotted to visualize the resulting efficiency and efficacy trade-off. We analyze each of these results in the following subsections.

4.1.1 Efficiency

Processing times listed in Fig. 4 were computed using an Intel i7 processor with 32 GB of RAM running an Ubuntu 20.04.2 LTS OS. Measurements clearly show that as the scenario complexity, i.e., the amount of nodes and contacts, increases, so does the time required to compute valid routes. However, the growth rate is much higher for CGR than for GAUSS. For instance, CGR takes 1.9 ms on average to deliver a route in scenario LEO A, while GAUSS is slower, taking 6 ms. However, as the topology becomes more extensive, as in scenario LEO D, CGR demands up to 1.2 s of computing time, while GAUSS delivers in half the time in the worst case. The trend is replicated and exacerbated in the cislunar scenarios, where GAUSS consistently delivers a route much faster than the CGR counterpart. Indeed, in Moon A and B with ISL (most extensive contact plans), we observed CGR calls requiring up to 2.5 and 2.7 seconds to complete, while GAUSS worst-case was 1.2 and 1.3 seconds. We also register that the Direct approach delivers the single-hop route immediately (1 ms). This is only valid for the Near-Earth scenarios. Of course, Direct is the best performing in terms of processing time. Still, it fails to find suitable paths in larger topologies where multi-hop is the only alternative to succeed in reaching a destination.

Regarding compute effort stability, CGR becomes more unpredictable and unstable in the expanded scenario LEO D, indicated by a flatter histogram with up to 1.1 seconds spread. On the contrary, GAUSS offers a more stable computing time (0.7 seconds spread), a fundamental feature for most space missions with tight operational constraints. This effect is also present in the cislunar cases, where the min/max spread of onboard routing effort is at the most 1.2 seconds for GAUSS but up to 2.7 seconds for CGR. Besides the min/max, the histograms in Fig. 4 clearly indicate that GAUSS compute time variability is significantly lower than CGR. This is confirmed by the standard variation metric indicated in the plots, which evidences that GAUSS onboard processing time is consistently closer to the mean than CGR.

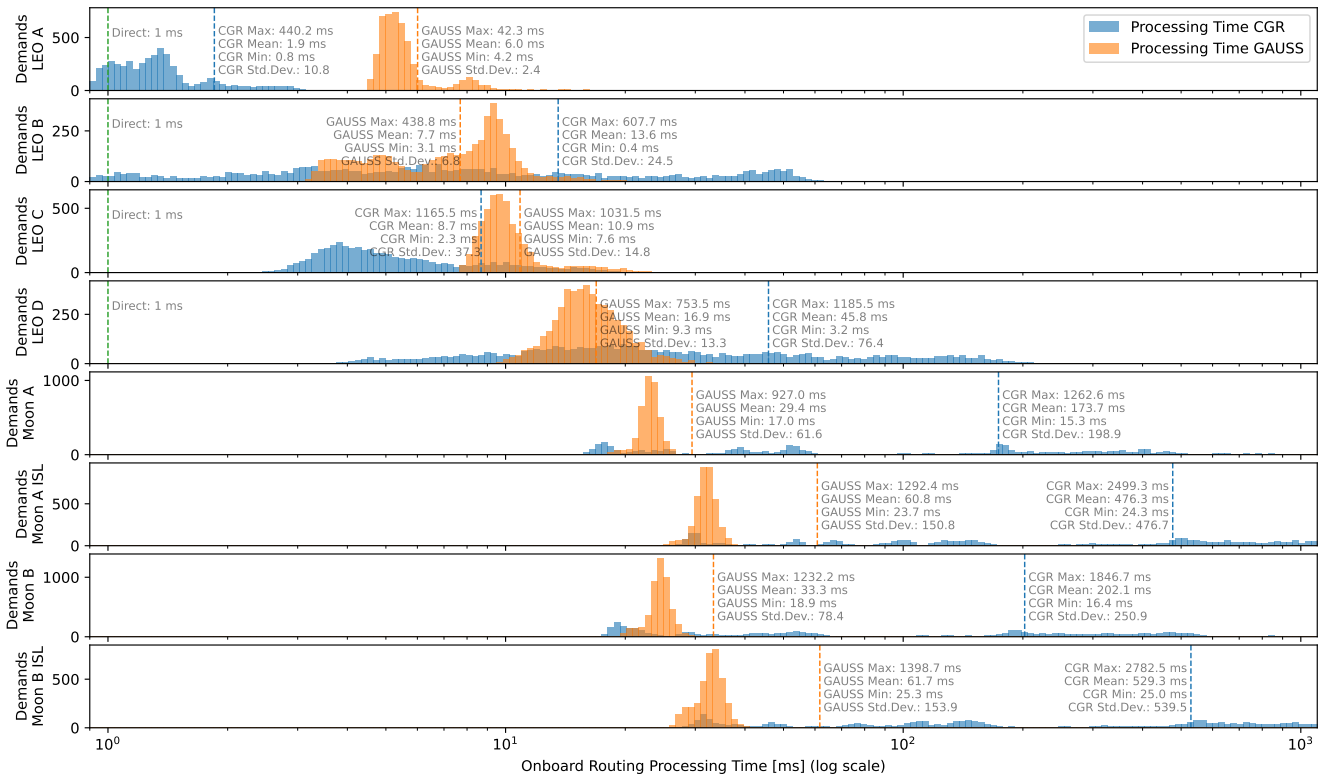

Figure 4: Efficiency results. Onboard routing processing times for Near-Earth (top) and Cislunar (bottom) scenarios.

4.1.2 Efficacy

It is fair to note that GAUSS's better computational effort profile comes at the expense of decreased efficacy. As listed in Table 2, and plotted in Fig. 5, the average delivery time of CGR is better, but the improvement increases as the contact plan grows. While in LEO A, the difference is minimal (3.3 seconds vs. 4.6 seconds), the trend deepens noticeably in scenario LEO D, where CGR is three times more efficient in promptly delivering data (4.9 seconds vs. 15.9 seconds). However, GAUSS still performs better than the *Direct* baseline, especially in scenario LEO C, where GAUSS is one-third times faster. The trend is less evident in the cislunar scenarios. Fig. 5 shows that GAUSS can keep up with CGR delivery time metrics for Moon A ISL and Moon B (approx. 20 seconds delivery time). However, GAUSS struggles to sustain metrics similar to CGR in scenarios Moon A and Moon B ISL (twice and four times worse than CGR). Nevertheless, note that it is in these cislunar scenarios where GAUSS processing gain was the highest, evidencing the efficiency-efficacy trade-off. Fig. 5 provides further insights into the analysis regarding the delivery ratio. This metric indicates the proportion of packets that were delivered to the destination. By observing this plot, the drawback of the *Direct* approach becomes evident in the cislunar scenarios, where topology partition forbids the existence of direct single-hop paths. Furthermore, the delivery ratio measures the end-to-end routes that GAUSS could not discover during training and inference.

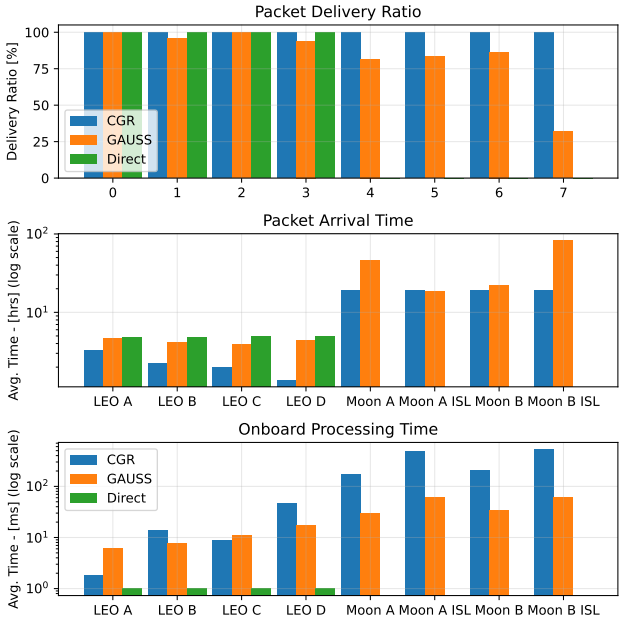

Figure 5: Efficacy results. Arrival time (top), delivery ratio (middle), and Processing time (bottom) for all scenarios.

As a wrap-up of the analysis, we have confirmed that GAUSS offers a new routing alternative for space DTNs with an appealing trade-off between resource efficiency and network performance. Specifically, GAUSS offers a scalable and stable routing solution from the processing perspective. Larger topologies can be accommodated while keeping the onboard variability for the routing compute routing. Because of these substantial advantages, GAUSS can become an attractive alternative to state-of-the-art CGR for space missions where processing effort must be reduced and stable. Of course, the approach is penalized with sub-optimal data delivery metrics. In any case, and supported by the evidence of scenarios Moon A ISL and Moon B, we believe there is room for fine-tuning the GAUSS model to fit specific space networks so that the delivery time can compete with CGR. This is left as an attractive future research direction.

5 Conclusions

In a context where space missions are growing in number and size, designing scalable and stable routing mechanisms with practical considerations is of primary importance. Supported by the latest advances in GNNs, we have developed GAUSS, a novel approach to route data flows in spaceborne DTNs. Evaluation results over realistic near-Earth space topologies and Lunar networks showed that onboard compute time can be reduced by half while reducing the effort variability spread to one-third. Because of this, we claim GAUSS will become an attractive candidate for resource-constrained space missions requiring consistent computing times to facilitate operations. In these cases, GAUSS outweighs the arrival time penalty when compared to the optimal routing alternative offered by CGR. Future work includes refining and improving the GAUSS model to bring network metrics closer to CGR. We also plan to implement prototypes in open-source DTN stacks like ION and μD3TN.

Acknowledgement

This research was supported by the European Union's Horizon 2020 R&D program under the Marie Skłodowska-Curie grant agreement No 101008233 (MISSION project).

6 References

[1] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, et al. Relational inductive biases, deep learning, and graph networks, 2018.

[2] S. Burleigh. Interplanetary overlay network: An implementation of the dtn bundle protocol. 2007.

[3] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss. Delay-tolerant networking: an approach to interplanetary internet. *IEEE Comms. Magazine*, 41(6):128–136, 2003.

[4] P. G. Buzzi, D. Selva, and M. S. Net. *Exploring Reinforcement Learning for Autonomous Delay Tolerant Network Management*.

[5] C. Caini, H. Cruickshank, S. Farrell, and M. Marchese. Dtn: An alternative solution for future satellite networking applications. *Proceedings of the IEEE*, 99:1980 – 1997, 11 2011.

[6] C. Caini and R. Firrincieli. Dtn for leo satellite communications. In G. Giambene and C. Sacchi, editors, *Personal Satellite Services*, pages 186–198, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[7] C. Caini and R. Firrincieli. Application of contact graph routing to leo satellite dtn communications. In *2012 IEEE ICC*, pages 3301–3305, 2012.

[8] V. Chourasia, S. Pandey, and S. Kumar. Optimizing the performance of vehicular delay tolerant networks using multi-objective pso and artificial intelligence. *Computer Comms.*, 177:10–23, 2021.

[9] R. Dalal and M. Khari. Peculiar effectual approach: Q-routing in opportunistic network. In *Int. Conference on Industrial Instrumentation and Control*, pages 609–615, Singapore, 2022.

[10] R. Dudukovich, A. Hylton, and C. Papachristou. A machine learning concept for dtn routing. In *2017 IEEE WiSEE*, pages 110–115, 2017.

[11] R. Dudukovich and C. Papachristou. Delay tolerant network routing as a machine learning classification problem. In *2018 NASA/ESA Conference on AHS*, pages 96–103, 2018.

[12] R. Dudukovich, K. Wagner, S. Kancharla, J. Fantl, and A. Fung. Towards the development of a multi-agent cognitive networking system for the lunar environment. In *2021 IEEE WiSEE*, pages 7–13, 2021.

[13] A. Elwhishi, P.-H. Ho, K. Naik, and B. Shihada. Arbr: Adaptive reinforcement-based routing for dtn. In *2010 IEEE 6th International Conference on Wireless and Mobile Computing, Networking and Communications*, pages 376–385, 2010.

[14] M. Feldmann, J. A. Fraire, and F. Walter. Tracking lunar ring road communication. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2018.

[15] J. A. Fraire, O. De Jonckère, and S. C. Burleigh. Routing in the space internet: A contact graph routing tutorial. *Journal of Network and Computer Applications*, 174:102884, 2021.

[16] J. A. Fraire and J. M. Finochietto. Design challenges in contact plans for disruption-tolerant satellite networks. *IEEE Communications Magazine*, 53(5):163–169, 2015.

[17] J. A. Fraire and E. L. Gasparini. Centralized and decentralized routing solutions for present and future space information networks. *IEEE Network*, 35(4):110–117, 2021.

[18] J. A. Fraire, P. G. Madoery, A. Charif, and J. M. Finochietto. On route table computation strategies in delay-tolerant satellite networks. *Ad Hoc Networks*, 80:31–40, 2018.

[19] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212, 2017.

[20] R. Hernández-Jiménez, C. Cardenas, and D. Muñoz Rodríguez. Modeling and solution of the routing problem in vehicular dtns: A dual, deep learning perspective. *Applied Sciences*, 9(23), 2019.

[21] T. Hu and Y. Fei. An adaptive and energy-efficient routing protocol based on machine learning for underwater delay tolerant networks. In *2010 IEEE MASCOTS*, pages 381–384, 2010.

[22] T. Hu and Y. Fei. An adaptive routing protocol based on connectivity prediction for underwater disruption tolerant networks. In *2013 IEEE GLOBECOM*, pages 65–71, 2013.

[23] J. Ji, H. Liu, A. Wang, and Y. Hao. Load forecasting-based congestion control algorithm for delay-tolerant networks. In *2020 SAGC*, pages 62–66, 2020.

[24] A. Lindgren, A. Doria, and O. En. Poster: Probabilistic routing in intermittently connected networks. *SAPIR Wrkshp*, 7, 06 2003.

[25] L. P. Portugal-Poma, C. A. Marcondes, H. Senger, and L. Arantes. Applying machine learning to reduce overhead in dtn vehicular networks. In *2014 Brazilian Symposium on Computer Networks and Distributed Systems*, pages 94–102, 2014.

[26] K. Scott and S. C. Burleigh. Bundle Protocol Specification. RFC 5050, Nov. 2007.

[27] D. K. Sharma, S. K. Dhurandher, et al. A machine learning-based protocol for efficient routing in opportunistic networks. *IEEE Systems Journal*, 12(3):2207–2213, 2018.

[28] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Spray and wait: An efficient routing scheme for intermittently connected mobile networks. In *2005 ACM SIGCOMM*, WDTN '05, page 252–259, New York, NY, USA, 2005.

[29] M. Sweeting. Modern small satellites-changing the economics of space. *Proceedings of the IEEE*, 106(3):343–361, 2018.

[30] L. Torgerson, S. C. Burleigh, et al. Delay-Tolerant Networking Architecture. RFC 4838, Apr. 2007.

[31] G. Wang, S. C. Burleigh, R. Wang, L. Shi, and Y. Qian. Scoping contact graph-routing scalability: investigating the system's usability in space-vehicle communication networks. *IEEE VTM*, 11(4):46–52, 2016.

[32] F. Yuan, J. Wu, H. Zhou, and L. Liu. A double q-learning routing in delay tolerant networks. In *ICC 2019 - 2019 IEEE ICC*, pages 1–6, 2019.