Towards graph-based machine learning for conjunction assessment

Emma Stevenson*
Universidad Politécnica de Madrid

Victor Rodriguez-Fernandez
Universidad Politécnica de Madrid

Hodei Urrutxua
Universidad Rey Juan Carlos

ABSTRACT

In the face of increasing space traffic, the deployment of large constellations, and a growing debris field, identifying potentially catastrophic collisions is an increasingly daunting and computationally challenging task. In this work, we present a novel graph-based machine learning approach for detecting conjunctions between catalogued space objects to aid in this task. Modelling conjunction events as edges between pairs of object nodes, we introduce a graphical representation of the all-vs-all scenario (so-called as it considers conjunction events between all catalogued objects, both active and debris) that is able to profit from recent advancements in Graph Neural Networks, and make a step towards efficient, machine learning based conjunction assessment. For this, we develop a methodology to predict the existence of upcoming conjunction links over a given screening period, which we frame as a graph-to-graph link prediction task, and present some initial findings that demonstrate the learning potential of the proposed approach.

1. INTRODUCTION

The prevention of in-orbit collisions is crucial both in the near-term, to protect current-day space assets, and for long-term space sustainability. To this end, identifying close approaches between all catalogued objects, whether active or debris, is of vital importance for early detection of potentially catastrophic events. However, this problem of all-vs-all conjunction assessment is computationally challenging, with hundreds of millions of possible conjunction pairs already present today, and threatens to become ever-more so in the face of increasing space traffic and observational capabilities in the New Space era [1].

With growing scales, data and demand for automation, the adoption of recent advancements in the field of machine learning (ML) has emerged as a promising research direction for tackling this problem [2], and for Space Situational Awareness (SSA) and Space Traffic Management (STM) at large [3]. Deep learning (DL) in particular, which uses multi-layer neural networks to train accurate, scalable predictive models, has been used to improve upon classical methods in both performance and efficiency in a wide variety of applications, from conjunction screening to risk assessment and manoeuvre planning [2]. In this paper, we build on previous works by the authors on leveraging these techniques for the task of all-vs-all conjunction screening, training a deep learning model to classify candidate conjunction pairs as an efficient first filter for reducing the computational burden on pre-existing operational conjunction assessment pipelines. These previous approaches have focused on *tabular* and *time series* based deep learning techniques using much the same formulation as traditional filtering approaches – by iterating over (batches of) individual object pairs.

In the all-vs-all case however, interactions (edges, or links) between pairs of individual objects (nodes, or vertices) over a whole catalogue can be more naturally described using a *graph*. Graph-structured data such as this is prevalent in a wide variety of different domains, from social networks, to transport networks, to molecular graphs. Following the recent successes of Graph Neural Networks (GNNs) in these areas with breakthrough models such as AlphaFold

*Email: emma.stevenson@upm.es

[4], the application of deep learning to graph-structured data has become one of the fastest growing research areas in machine learning [5]. However, the flexibility gained in loss of structure is coupled with greater learning challenges. For one, unlike for grid-like data, such as the aforementioned sequences or images, which have a standard data representation, the step of building a representation of the input graph that can be successfully exploited for graph tasks such as link prediction (e.g., recommending new social network connections, or predicting conjunctions between two space objects) is no longer trivial.

Thus far, the intersection between graph and network theory and the space domain has been primarily restricted to topological modelling of the space environment with a view to studying its longer-term evolution and stability [6, 7, 8, 9]. By studying statistical properties of networks constructed of dynamically varying space object nodes and conjunction edges, these works seek to understand the resilience of the space environment to potential Kessler Syndrome-triggering (cascading) events, and to quantify the impact of different mitigation strategies and Active Debris Removal missions on this overall collision risk. By contrast, in this work we make the first steps towards exploring the usability of such a graph representation for the shorter, typically week-long timescales associated with conjunction assessment activities, in the context of GNNs.

As such, in this paper we present a novel graph-based, global representation of the all-vs-all scenario that is able to profit from recent advancements in GNNs, and make a step towards efficient, machine learning based conjunction assessment. Focusing on the task of conjunction screening, we investigate the suitability of graph representations and constructions for detecting close encounters, which we frame as the prediction of conjunction links between space object nodes over a given screening period. To the best of our knowledge, this is the first time graph-based machine learning algorithms have been applied in this way in the space domain.

The paper is structured as follows. First, in Section 2, we provide relevant backgrounds on graph-based machine learning tasks and architectures, as well as on operational conjunction assessment and the classical filters used therein. Then, in Section 3, we describe our proposed approach, providing a detailed discussion on the choice and method of construction of the graph task, graph representation and training data set. Here, we also detail an array of learning considerations that we found to be necessary to successfully train a graph-based model for the task of conjunction detection, before introducing some preliminary experimental results in Section 4. Finally, in Sections 5 and 6 we highlight several important open aspects and promising avenues for future work, and summarise the main conclusions of the paper.

2. BACKGROUNDS

In this Section, we first provide background details on the two worlds explored in this paper: graph-based machine learning including relevant learning tasks and architectural components of graph neural networks (see Section 2.1); and a brief review of current operational conjunction assessment with a focus on conjunction screening (see Section 2.2).

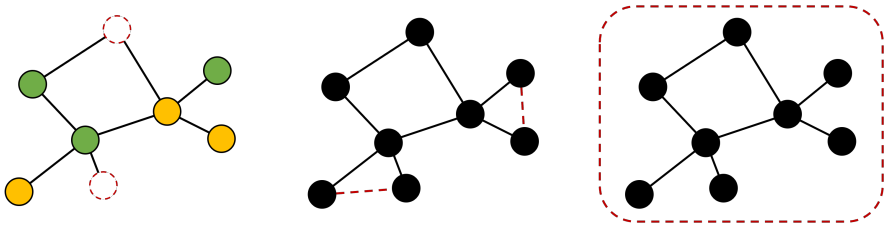
2.1 Graph-based machine learning

A graph $G = (V, E)$ is a mathematical structure used to model pairwise relations between objects. Objects are represented as nodes (or vertices $v \in V$) and their relationship is represented by edges (links or arcs) connecting two nodes, i.e., by pairs $(u, v) \in E$ with $u, v \in V$. Edges can be directed, meaning that they have a direction associated with them, or undirected, which can be traversed in both directions. Internally, the most common representation of a graph employs the so-called adjacency matrix, whose position (i, j) gives the presence of an edge between nodes i and j, i.e., whether $(i, j) \in E$. However, nodes and edges can also have attributes (or features), which are not directly related to the graph itself, but rather to properties of the node or the edge.

Machine learning on graphs is an important and ubiquitous field with applications ranging from drug design to friendship recommendation in social networks [5]. Below are some of the most popular learning tasks belonging to this area, for which a graphical description is given in Fig. 1:

- *Node classification*, where the model predicts node properties (or labels) based on information of the node itself and its surrounding nodes in a single graph (See Fig. 1a).
- *Link prediction*, which is the task of estimating the probability of missing or future edges between nodes in a single graph, given its current state (See Fig. 1b).

- *Graph classification*, which is the task of assigning labels to a whole new graph, given a dataset of multiple graphs (See Fig. 1c).



(a) Node-level (b) Edge-level (c) Graph-level

Fig. 1: Graph learning tasks can typically be divided into: (a) node-level (node classification, regression and clustering); (b) edge-level (link prediction); (c) graph-level (graph classification and regression).

Traditional graph-based machine learning relied on user-defined heuristics to extract features encoding structural information about the graph [5]. However, due to the recent outbreak of neural networks and deep learning as a de facto solution for many machine learning tasks in areas such as computer vision and natural language processing, the research community of graph-based machine learning has shifted towards the use of Graph Neural Networks (GNNs), and in fact, as part of this, a new field of research, known as geometric deep learning, has recently emerged, as a discipline that uses deep neural networks on data with an underlying non-Euclidean structure [10].

Graph neural networks are deep neural networks for machine learning problems on graphs. Unlike standard neural networks, GNNs retain a state that can represent information from the node neighborhood with arbitrary depth, and they incorporate that state in the learning process. To do so, a process known as *message passing* is employed, in which neighbour nodes send messages iteratively to each other and update their state [11] with the information of those messages. GNN models aggregate messages for each node from its neighborhood in the previous layer.

In terms of architecture, a GNN is typically built by combining three computational modules or steps:

- Propagation module: used to propagate information between nodes so that the aggregated information can capture both feature and topological information of the graph.
- Sampling module: Needed only for large graphs in order to reduce the size of neighboring nodes and alleviate the so-called "neighbor explosion" issue during propagation.
- Pooling module: It extracts high level information from the features captured in the propagation module.

There is a plethora of implementations for each of the aforementioned modules, especially in the case of the propagation step[1]. Here, we can find graph-based versions of the most common layers (or operators) used to process images and text, such as graph convolutions [12], which generalise convolutions out of grid-like data, or graph attention [13], which leverages the self-attention mechanism that was popularised in Natural Language Processing with the Transformer architecture [14], in such a way that the state of each node is computed by "attending" to its neighbours. The use of these operators in the propagation module gives name to the network as a whole, and therefore it is common to see references to Graph Convolution Networks (GCNs) and Graph Attention Networks (GATs).

2.2 Conjunction assessment

Operational conjunction assessment and collision avoidance activities are responsible for preventing the destruction of space assets and proliferation of space debris through collisions, and are typically divided into the following 3-step procedure [15]:

1. Conjunction screening: identification of close approaches between catalogued space objects by comparing object trajectories and determining when and where collisions may occur.

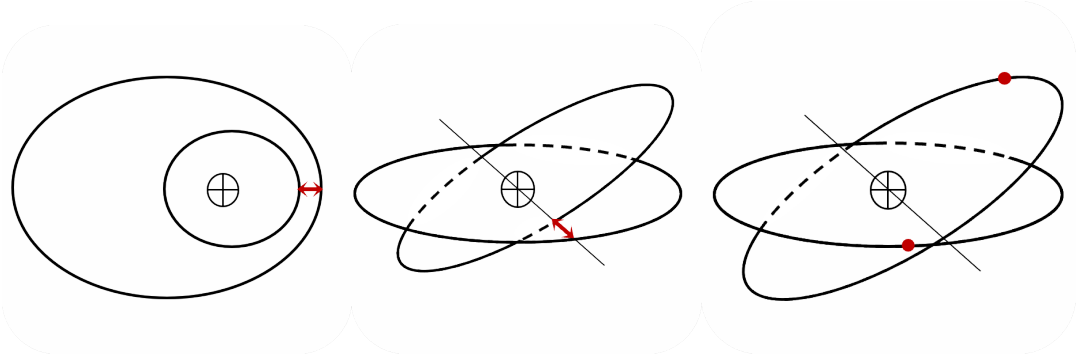
[1] A list of some of them can be found in `https://pytorch-geometric.readthedocs.io/en/latest/notes/cheatsheet.html`

2. Risk assessment: evaluation of probability of collision of identified conjunction events.

3. Conjunction mitigation: planning and execution of a Collision Avoidance Manoeuvre (CAM) to reduce the collision risk of given events to an acceptable level.

To reduce the computational burden associated with this procedure, by way of processing a smaller number of candidate conjunction pairs, operators today typically only consider possible conjunctions between their own fleet of satellites and the full space object catalogue (the *one-vs-all* scenario) in the initial screening step. This approach circumvents the true quadratic scales associated with the *all-vs-all* problem, whilst ensuring that any output is actionable from the perspective of the operator (i.e. they may actually perform a CAM to protect their asset). However, the importance of the often-neglected and vast number of remaining possible debris-debris collisions should not be overlooked. Fragment clouds such as that generated by the infamous 2009 Iridium 33 - Cosmos-2251 event can be an operational threat for many decades [16], and thus it is crucial for space situational awareness and future space traffic management to also be able to process and identify these events.

Fundamentally, identifying close approaches can be achieved by comparing pairs of trajectories step-wise over a given screening period to determine whether (and when) they pass within a given safety volume. However, this process can be laborious, especially in the all-vs-all scenario, and thus to expedite it, a series of successive filters can first be applied to quickly discard object pairs for which there is no possibility of an encounter. Such pairs can be identified using geometrical considerations, most commonly captured by the following series of filters [17]:



(a) Apogee-perigee (b) Path (c) Time

Fig. 2: Classical filters for conjunction screening.

1. Apogee-perigee: filters pairs whose orbits do not intersect based on orbit altitude (See Fig. 2a). Given the apogee, h_a, and perigee, h_p, altitudes of each object in a candidate conjunction pair, the pair may be discarded if, for a given threshold distance D,

$$\max(h_{p1}, h_{p2}) - \min(h_{a1}, h_{a2}) > D \tag{1}$$

2. Path-to-path: filters pairs whose orbits do not come closer than D, regardless of the position of the objects along their orbit (i.e. whose minimum orbital intersection distance, MOID $> D$) (See Fig. 2b).

3. Time: filters pairs accounting for the position of the objects along their orbit - even if the orbit planes intersect, the objects must pass through these orbit crossing points simultaneously for a conjunction to occur (See Fig. 2c).

Although these filters improve efficiency, it should be noted that they can lead to missed conjunctions owing to their basis in Keplerian dynamics and thus should be treated with care. Of the three, the apogee-perigee has been shown to be the most conservative and effective [18]. Once the set of reduced candidate conjunction pairs has been obtained, and the time of closest approach (TCA) of their associated events determined, they can then be passed to step 2 of the pipeline to incorporate uncertainty information in evaluating the probability of collision, using methods such as [19].

3. METHODOLOGY

In this Section we present our proposed methodology for graph-based machine learning in the task of conjunction detection, enabling us to move away from pairwise iteration in filtering to a more natural graphical representation of the all-vs-all collision scenario and space environment as a whole. We describe in detail how we construct the graph representation and learning task, including how we arrived there and why other approaches might not be possible, and detail a large number of learning considerations that should be taken into account when endeavouring to train a GNN for this task.

3.1 Graph construction

The basis of a screening problem is a catalogue comprising N space objects, each with an associated ephemeris propagated over a time period T, over which we would like to screen for possible conjunctions. As such, for a given input catalogue, we would like to obtain a series of output pairs which we determine to be "risky" over $t \in [t_0, T]$ and thus warrant further investigation for potential mitigative action.

In a graph context, each catalogued object may be naturally modelled as a node, with potentially dangerous interactions (conjunctions) modelled as edges. Intuitively, it follows that the learning task is one of *link prediction* (see Section 2.1), in which we train a model to predict upcoming conjunction links between the given nodes. Classically however, link prediction tasks are performed using a single graph (see Fig. 1b), in which the model is trained using a subset of edges that have ground-truth labels to infer other edges (for example, to recommend connections between users in a social network). Such an approach cannot be applied in our use case, in which we would like to feed the model with an input catalogue and subsequently predict the existence of any and all edges over that single graph. By predicting the existence of each edge (in other words a binary conjunction/no-conjunction label), the output of the model is simply the adjacency matrix and thus the machine learning task is that of *graph-to-graph* link prediction.

In graph-to-graph link prediction tasks, edges associated with the input graph may represent different relationships or have different properties/attributes to those in the output graph to facilitate the learning process. An example of a task that has been modelled successfully in this way is image scene understanding, which uses GNNs to predict connections between objects in an image [20]. In this case, the input is a fully connected graph, with each node representing a different object or pixel with the image. The output graph then prunes and labels these edges to uncover possible connections.

In a similar way, we can construct our input graph by linking nodes that may be connected (in the form of a conjunction), which can then be pruned or further filtered by the model. This allows our graph-based model to function much like one of the classical filter layers described in Section 2.2.

The construction of the input and output graphs is illustrated in Fig. 3. The input (catalogue) graph is defined by a set of N nodes, each with a set of attribute information. This consists of orbit information such as propagated ephemeris and associated time encodings, alongside any physical properties related to the space objects that may aid the learning process such as size, object type or catalogue ID. Node attributes may also include feature information extracted from the ephemeris by a backbone time series model. We refer interested readers on this topic to [21]. We then determine which nodes are connected through an edge using the apogee-perigee filter (Equation 1, see Section 2.2). This allows us to efficiently and reliably discount "impossible" pairs with large orbit separation, whilst retaining enough edges for the GNN to accurately encode contextual information on how the object nodes interact within the graph. Finally, we may assign weights or attributes to the edges themselves. These can be based on the apogee-perigee filter itself or other distance based metrics which give an indication of the degree of closeness of a given pair, which we discuss further in Section 3.3. In previous works, these link weights have been assigned using the average difference in apogee and perigee [8] or using the probability of impact during a conjunction [9], but different metrics or even time series of metrics may be employed depending on the graph architecture.

The output (collision) graph is then defined by the same nodes through a predicted $N \times N$ adjacency matrix. In this, the model aims to filter the apogee-perigee-based edges in the catalogue graph to only those resulting in a conjunction (as defined by the training data), with the aim of reducing the number of risky pairs that need further evaluation in the conjunction assessment pipeline. In both graphs, we consider that the edges are undirected, and thus the adjacency matrices are symmetric. It should also be noted that Fig. 3 shows a graph representation without self-loops (i.e. edges that connect a node to itself), resulting in a label of 0 (no-conjunction) on the leading diagonal.

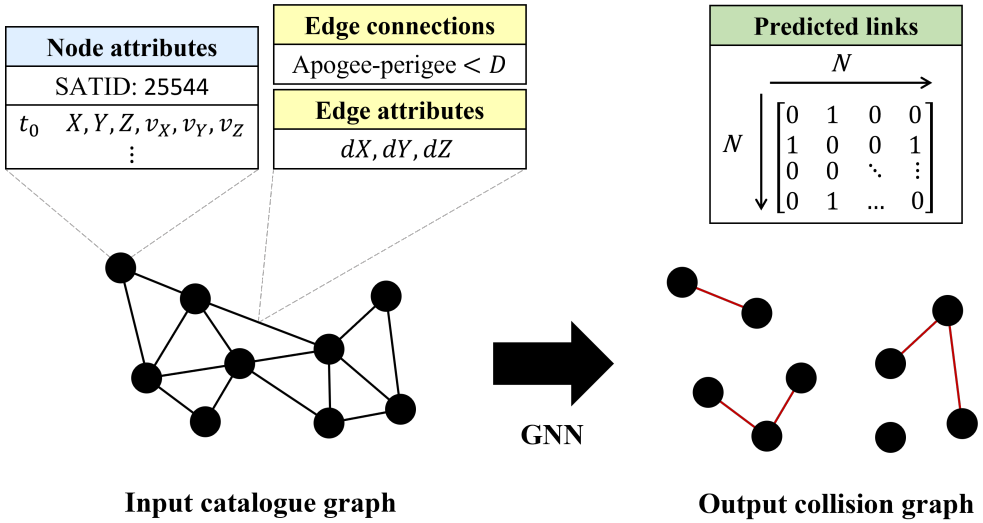


Fig. 3: Graph representation as a graph-to-graph link prediction task. Each node represents a catalogued object. In the input graph edges represent connections based on the apogee-perigee filter. Edges in the output graph represent predicted conjunctions.

3.2 Training set construction

To train such a graph-to-graph model, we need a data set of numerous input-target graph pairs. This could be achieved by breaking the catalogue down into subgraphs (although this approach would hinder our eventual aim of performing inference on the whole catalogue), or by building a data set from snapshots of the catalogue as it evolves in time. This latter approach is also more appropriate as in reality, graph inference will always be done in the future (from the perspective of the model) and thus our problem has a temporal dimension. That is to say, we would like to deploy our model to predict conjunctions over the next X days, given that the model will have been trained on past ephemerides and edge labels. The model must therefore have enough astrodynamical understanding to generalise well in time, which can be introduced through a greater variety of past orbits using a time series of graphs.

We construct our data set using a sliding window approach, as illustrated in Fig. 4. Given a labelled data set of conjunctions over a given time period, we can construct a series of graphs covering different sub-time windows and therefore different conjunction events. The window stride may also be chosen such that these windows overlap in order to maximise the number of graphs generated from the available labelled data. For each window, the input and target graphs are constructed following the description in Fig. 3.

Finally, the data set is split into training and validation subsets, a standard practice in machine learning for evaluating the performance and generalisation ability of the model during training using an independent data set. This can be used to identify if the model is over-fitting (memorising) the training data, as well as to tune higher level hyperparameters that are pre-set by the user and not learnt during training, such as architectural parameters. Due to the correlated nature of time series data, a nominal random splitting strategy cannot be used and thus we adopt a splitting approach used in the domain of time series forecasting in which we assign the final graphs of the time series to the validation set, ensuring that their time windows do not overlap with those used for training to prevent data leakage (see Fig. 4).

3.3 Learning considerations

We implement this data set design and the subsequent training procedure using the PyTorch Geometric (PyG) library [22][2]. PyG hosts a range of state of the art graph neural network architectures and methods for deep learning on graph and geometric structured data and is built, as the name suggests, on top of the python-based PyTorch deep learning framework. To profit from these, we construct our data set for the learning task of graph-to-graph link prediction by building each graph as follows:

[2]`https://github.com/pyg-team/pytorch_geometric`

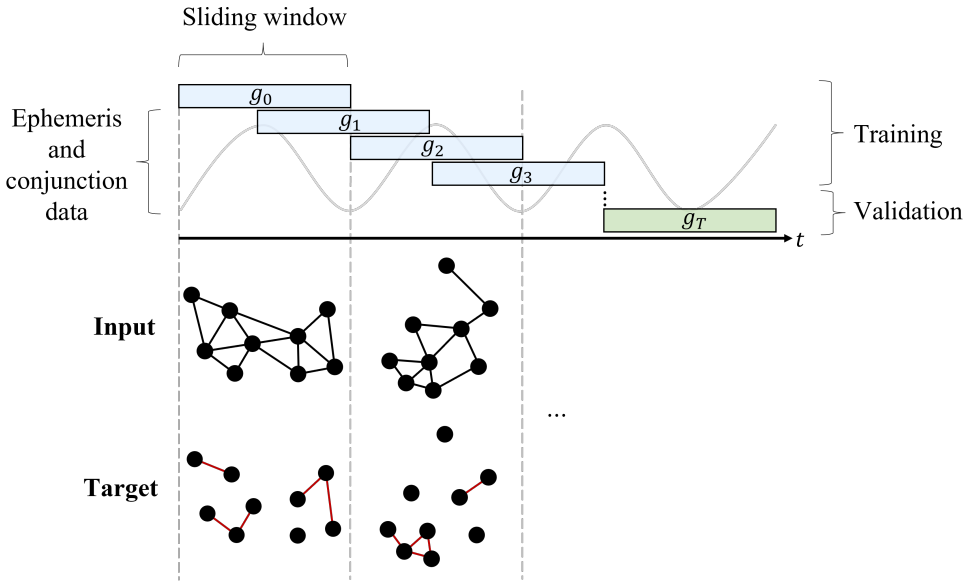


Fig. 4: Training data set comprises a time series of graphs, generated using a sliding window approach.

- Input graph node attributes (X) - for each object node in the catalogue graph, we include a set of features capturing trajectory information that the model can interpret for determining close approaches. We can consider a variety of different features and representations based on the object ephemeris to aid in this task including: the number of states (frequency of the time series); the state representation (for example, a state vector representation, associated with the position and velocity vectors of each object, classical orbital element, equinoctial or a combination approach); encodings for the state epochs to encourage better generalisation of the model in time (for example, seasonal or periodic encodings to help the model capture patterns associated with different perturbative effects).
- Input graph edge connections (E_{index}) - the nearest neighbours of each given node (as determined using the apogee-perigee filter for a specified threshold radius D) are connected via undirected edges in a symmetric adjacency matrix. For memory efficiency, this (relatively sparse) matrix is stored as an adjacency list which, to avoid storing large quantities of zero elements between disconnected parts of the graph, only stores indexed connected edges as a set of tuples.
- Input graph edge attributes (E_{attr}) - weights or series of features associated with each edge connection. May be based on relative positional information between connected nodes, or distance based metrics such as the apogee-perigee filter used to construct the connections, but choice should be computationally efficient and scalable (i.e. not requiring too many pairwise computations), and effectiveness may be coupled with choice of graph architecture (see discussion below).
- Target graph labels (y) - $N \times N$ adjacency matrix with edge connections denoting possible conjunctions (symmetric). This will be a sparse matrix (even more so than the input graph adjacency matrix), especially considering that some regions of the space object catalogue are less populated than others.

Having constructed the series of graphs, we split them into training and validation subsets as discussed in Section 3.2. We then normalise the data to account for differences in units and scales of the various features and attributes by transforming the mean of each feature to 0 and standard deviation to 1. This must be done using global normalisation factors found by considering all graphs (else the model cannot learn from how the graphs evolve in time in order to generalise to future epochs), which must be found using only the training data set to prevent data leakage.

The next important learning consideration concerns how the graph data can be *batched*. As in other realms of machine learning, it is advantageous not to feed a neural network with all the training data at once, but instead to sample random partitions of disjoint batches for each training epoch. This avoids memory constraints, but also increases stochasticity

which can aid optimisation during the learning process, and has been shown to provide improved generalisation performance. In the case of graph learning, different graph tasks require different batching approaches as they operate on two ends of the scale spectrum: using many small graphs (typically for graph-level tasks) or single giant graphs (node or edge-level tasks). For the first case, batches can be created by stacking the adjacency matrices of several smaller graphs diagonally, thereby building a giant graph containing several isolated subgraphs. This approach allows many smaller graphs to be processed simultaneously whilst ensuring that nodes from different subgraphs cannot communicate by message passing. On the other end of the spectrum, where the adjacency matrix is too large to be stored in memory, batching can be achieved by selecting a subset of nodes and then randomly subsampling neighbours, and neighbours' neighbours, as a subgraph for training [23]. This method can be used to handle extremely large graphs that have on the order of a billion nodes. As our graph task lies somewhere in between, comprising multiple potentially large graphs, the choice of batching approach depends on the size and scale of the catalogue being considered.

One further consideration is then whether to shuffle the order of the graphs during the batching procedure prior to each training epoch. This practice is typically harmful in time series tasks due to the sequential nature of the data, but was found here to improve consistency over batches during training, and to give more robust results, likely due to the use of explicit time encodings.

Next, we can consider which GNN architectures may be most appropriate. This choice is dependent on a variety of factors, including whether we wish to consider the graph as *static* or *dynamic*, as well as on attribute-related properties of the graph, with only some architectures supporting message passing with multi-dimensional edge attribute information (i.e. beyond a one-dimensional weight). Most available architectures have thus far been developed for static graphs, however, some architectures have recently emerged for learning on dynamic graphs, whose number of nodes and edges evolve over time (useful for tasks like future edge prediction - what is the probability of having an edge between two nodes i and j at time t?) [24]. Temporal information can also be captured from static graphs using a hybrid approach which combines time series models for extracting temporal features from node attributes, with GNNs for extracting node neighbourhood information [25].

In this preliminary work, we focus on building from a simpler modelling approach using a static representation, in which the nodes and edge connections of each graph (each screening period) can be considered to be static (i.e. the conjunction label does not change over the course of a single graph). For this graph and task representation, we found most success with the "SplineCNN" architecture [26], which uses a trainable spline-based convolution operator to learn from graph-structured data. This convolution filter is used to aggregate node feature information in local neighbourhoods using a weighting determined by a trainable kernel function. Whether the nodes can be considered local is determined by relative positional information, which is stored in coordinate form as the edge attributes (for existing edges, which we choose to represent in normalised Cartesian form).

Finally, we choose the loss function for our task to be the binary cross-entropy, as is typical of binary classification tasks, also integrating the sigmoid layer that maps the network output to class probabilities into the loss (*BCEWithLogitsLoss*) for improved numerical stability. As the target matrix y is symmetric, we only apply this loss to elements above the leading diagonal. Extending this to the leading diagonal and including self-loops (self-conjunctions) was also found to be beneficial for determining if the model was behaving as expected (intuitively, comparing nodes with identical trajectories should result in a conjunction).

4. PRELIMINARY RESULTS

In this Section, we aim to demonstrate that GNNs have the potential to learn from the graph construction developed in this paper for the task of conjunction detection. We do this by considering the ability of the model to *overfit*. Overfitting occurs when the model performs well on training data but fails to generalise well to unseen examples (for example the validation or testing set) and is typically one of the major challenges to overcome when training a deep learning model. However, its value as a first analysis step is usually overlooked when in fact it can be a useful tool to give an indication that the model is able to capture the necessary level of detail and noise in the training data, and detect meaningful patterns.

For these initial experiments we use a conjunction dataset that was generated using the CNES BAS3E (**B**anc d'**A**nalyse et de **S**imulation d'un **S**ysteme de **S**urveillance de l'**E**space – Simulation and Analysis Bench for Space Surveillance System) space surveillance simulation framework [27] for all catalogued object pairs in LEO over a 7-day screening period. For this, we considered the two-line element set (TLE) LEO population on the 1st of June 2020, as retrieved

from space track, which comprised 18415 objects, and consequently 170 million possible conjunction pairs. These objects were propagated over a 7-day screening period using a full force model, accounting for atmospheric drag and Sun and Moon third body perturbations, and checked pairwise for conjunctions using a distance based threshold of 20 km (spherical shape of the safety volume). More details on this data set may be found in [2].

Following the methodology outlined in Section 3 above, we construct the data set as follows. For initial testing we consider a small subset of the space object catalogue with perigee and apogee altitudes between 800 and 820 km, comprising 39 object nodes, all with at least one conjunction. Using a sliding window of size 1 day and stride 1 day (i.e. no overlapping windows) we generate a dataset of 7 graphs for these nodes (i.e. each graph snapshot considering conjunction events over a 24 hour period). For each node of each graph, we include the first and last state over the considered window, using both a state vector and orbital element representation. As the chosen architecture requires a 1D feature tensor, we concatenate these two states, obtaining 26 features for each node (i.e. $2 \times [t, X, Y, Z, V_X, V_Y, V_Z, a, e, i, \omega, \Omega, M]$). These nodes reside within a very small orbital neighbourhood and thus we subsequently use an atypically low value for the apogee-perigee threshold, D, of 100 m to ensure the input catalogue graphs are not fully connected, assigning the input edge connections per graph based on the initial state. Although the number of edges varies per graph, with this approach, the graphs are approximately 80% connected. The attributes of these edges are then assigned as the positional differences (i.e. $[dX, dY, dZ]$) between adjoining nodes again using the initial state. Finally, we split the data set into 6 graphs for training and 1 for validation, and normalise over the 26 node features using the whole training set, as discussed in Section 3.3.

To train the model we used the spline-based convolutional architecture SplineCNN with 6 convolutional layers, 64 hidden channels and a kernel size of 5, forcing the output channels to have a size of the number of input nodes (39). As these are small graphs, we are able to use a single batch for training (see discussion on batching in Section 3.3), ensuring that we shuffle the order of the graphs before each epoch. We then train the model for 500 epochs with the Adam optimiser, initialised with a learning rate of 0.001.

In Fig. 5 we show an example of the predictions of the resulting model on one of the conjunction graphs used for training, alongside the true labels. Conjunctions are shown in black, and only for the upper triangle which is considered for the loss. Here, we also consider self-conjunctions (i.e. the leading diagonal also comprises conjunction events) to ensure the model was behaving correctly – although these will be removed in future work so these events do not overshadow the true conjunction events. It can be seen that the model is successfully able to train and memorise the training data, thus demonstrating that the proposed graph and task construction has the potential to profit from a GNN approach to conjunction screening.

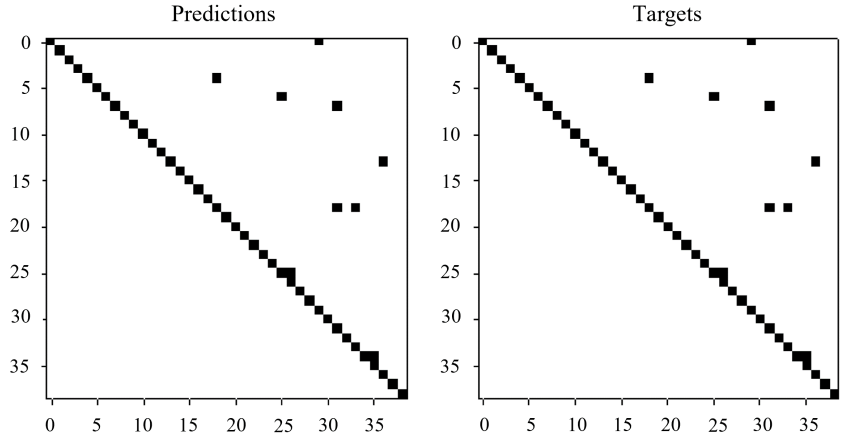


Fig. 5: Predicted and target conjunction events (black, including self-conjunctions) for an example training graph of 39 object nodes to demonstrate the ability of the model to overfit and memorise the input training data.

At present the model does not show a similar level of performance when generalising to the validation set, the next crucial step, but we theorise that this difficulty arises as we try to generalise in time (i.e. to a distribution outside that which we have trained on) and thus more data and further experimentation are necessary.

5. OPEN ASPECTS

Although the approach proposed in this paper represents an important first step in moving towards graph-based conjunction assessment, we highlight here a number of open aspects and possible deficiencies in this initial iteration which we hope will pave the way for future research in this field:

- *Scaling* up the proposed approach to work with an entire catalogue is challenging due to, among other things, the output collision graph growing quadratically with the number of nodes in the input graph. However, due to the significant variation in object density between different altitude bands and the lack of high risk collisions between them, that graph is sparse (full of zeros) in nature. Therefore, relying on graph *sparsity* implementations would be helpful to improve the scalability of our methodology. Another aspect that would alleviate scalability issues is to re-frame the proposed prediction task, so that only the top-M riskiest collisions are predicted for each of the N objects of the catalogue. This would result in an output matrix of size $N \times M$ instead of $N \times N$, with $M << N$, which scales linearly instead of quadratically. However, such a matrix would not be a nominal binary adjacency matrix, and more research is needed in terms of how to implement such an approach.
- The proposed approach fails in its ability to adapt to changes in the object catalogue, especially in reference to the *addition/deletion of objects*. As the output collision graph is a $N \times N$ matrix, a GNN trained on a given catalogue with N objects would only be usable with an identical number of catalogued objects. Changing the size of the output matrix in this approach would require either retraining the neural network from scratch, or at least fine-tuning it with every catalogue object update, which would be infeasible from an operational perspective. This would be further complicated by the inclusion of static node attributes such as satellite ID and physical properties.
- Graphs are flexible data structures in which nodes do not have an explicit position within a space. Although these are entirely suitable for representing non-physical entities such as users in a social network, modelling real-world spatiotemporal data such as the evolution of a space catalogue may benefit from data structures that explicitly encode spatial information, such as point clouds.

6. CONCLUSIONS

This exploratory paper of two worlds presents a novel graph-based machine learning approach for detecting conjunctions between space objects. Graph structured data is arguably the most natural representation for the ever-growing all-vs-all problem, in which conjunctions can be modelled as edges between pairs of individual object nodes over the space object population. However, despite the recent surge of successes in the realm of Graph Neural Networks (GNNs), constructing and learning from these flexible data structures is a challenging task.

In a first application of GNNs to this problem, this work is dedicated to investigating the suitability of different graph tasks and constructions for detecting close encounters, in which we aim to predict conjunction links between space object nodes over a given screening period. We arrive to the task of graph-to-graph link prediction, in which candidate conjunction links in an input catalogue graph (with ephemeris data associated to each node and edge connections built using a nearest neighbours approach based on the classical apogee-perigee filter) are pruned by the GNN, returning an output graph with predicted conjunctions between the same set of nodes. This custom task formulation is further complicated by the scales and time-evolving nature of the data, and thus we also detail a large number of learning considerations which are important to the feasibility of exploiting GNNs for this problem, such as construction of the training data set, batching, and graph architecture. Finally, we demonstrate the potential of our proposed approach by successfully training and overfitting a GNN, and present several open aspects to pave the way for future graph-based SSA and STM applications.

ACKNOWLEDGMENTS

This research is supported by the EU H2020 MSCA ITN Stardust-R, grant agreement 813644[3] and the Spanish State Research Agency and the European Regional Development Fund through the research grant PID2020-112576GB-C22

[3]Stardust Reloaded: `https://doi.org/10.3030/813644`

(AEI/ERDF, UE). The authors would like to gratefully acknowledge Vincent Morand for his help and technical support in providing the BAS3E dataset.

REFERENCES

[1] T. J. Muelhaupt, M. E. Sorge, J. Morin, and R. S. Wilson. Space traffic management in the new space era. *Journal of Space Safety Engineering*, 6(2):80–87, June 2019.

[2] E. Stevenson, V. Rodriguez-Fernandez, H. Urrutxua, V. Morand, and D. Camacho. Artificial intelligence for all vs. all conjunction screening. In *Proceedings of the 8th European Conference on Space Debris*, (Virtual), Darmstadt, Germany, 2021.

[3] L. Sánchez Fernández-Mellado, M. Vasile, and E. Minisci. AI and Space Safety: Collision Risk Assessment. In *Handbook of Space Security: Policies, Applications and Programs*, pages 1–19. Springer International Publishing, Cham, 2020.

[4] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, August 2021.

[5] W. L. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 66:257–268, 2017.

[6] H. G. Lewis, R. J. Newland, G. G. Swinerd, and A. Saunders. A new analysis of debris mitigation and removal using networks. *Acta Astronautica*, 66(1):257–268, January 2010.

[7] R. J. Newland, H. Lewis, and G. Swinerd. Supporting the development of active debris removal using weighted networks. In *Proceedings of the 5th European Conference on Space Debris*, Darmstadt, Germany, 2009.

[8] G. Acciarini and M. Vasile. A multi-layer temporal network model of the space environment. In *Proceedings of the 71st International Astronautical Congress*, Cyberspace Edition, 2020.

[9] M. Romano, T. Carletti, A. Lemaitre, and J. Daquin. Assessment of uncertainty propagation techniques for the network embedding of the space resident population. In *Proceedings of the 5th International Workshop on Key Topics in Orbit Propagation applied to Space Situational Awareness (KePASSA)*, Logroño, Spain, 2022.

[10] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[11] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, January 2020.

[12] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017*, 2017.

[13] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[15] NASA. NASA spacecraft conjunction assessment and collision avoidance best practices handbook. NASA/SP-20205011318, December 2020.

[16] ESA Space Debris Office. ESA's Annual Space Environment Report. issue 6.0. GEN-DB-LOG-00288-OPS-SD, ESA/ESOC, Darmstadt, Germany, April 2022.

[17] F. R. Hoots, L. L. Crawford, and R. L. Roehrich. An analytic method to determine future close approaches between satellites. *Celestial mechanics*, 33(2):143–158, June 1984.

[18] J. Woodburn, V. Coppola, and F. Stoner. A description of filters for minimizing the time required for orbital conjunction computations. In *Proceedings of the AAS/AIAA Astrodynamics Specialist (AAS) Conference*, Pittsburgh, Pennsylvania, United States, 2009.

[19] K. T. Alfriend, M. R. Akella, J. Frisbee, J. L. Foster, D. Lee, and M. Wilkins. Probability of collision error analysis. *Space Debris*, 1(1):21–35, March 1999.

[20] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 2021. https://distill.pub/2021/gnn-intro.

[21] E. Stevenson, V. Rodriguez-Fernandez, H. Urrutxua, V. Morand, and D. Camacho. Self-supervised machine learning based approach to orbit modelling applied to space traffic management. In *Proceedings of the 11th International Association for the Advancement of Space Safety (IAASS) Conference*, (Virtual), Rotterdam, The Netherlands, 2021.

[22] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, New Orleans, Louisiana, United States, 2019.

[23] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *31st Conference on Neural Information Processing Systems (NIPS)*, Long Beach, California, United States, 2017.

[24] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv:2006.10637*, October 2020.

[25] E. Rocheteau, C. Tong, P. Veličković, N. Lane, and P. Liò. Predicting Patient Outcomes with Graph Representation Learning. *arXiv:2101.03940*, January 2021.

[26] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller. SplineCNN: Fast geometric deep learning with continuous B-spline kernels. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, Utah, United States, 2018.

[27] V. Morand, C. Yanez, J. C. Dolado Perez, C. Fernandez, S. Roussel, X. Pucel, and V. Vidal. BAS3E: A framework to conceive, design, and validate present and future SST architectures. In *Proceedings of the 1st NASA International Orbital Debris Conference*, Houston, Texas, United States, 2019.