Counterfactual Explanations for Black-Box Models on Dynamic Graphs

Master Thesis

by

Daniel Gomm

Degree Course: Industrial Engineering and Management M.Sc.
Matriculation Number: 2055065

Institute of Applied Informatics and Formal Description Methods (AIFB)

KIT Department of Economics and Management

Advisor:	Dr. Michael Färber
Second Advisor:	Prof. Dr. Andreas Oberweis
Supervisor:	M.Sc. Zhan Qu
Submitted:	December 12, 2023

Abstract

Temporal Graph Neural Networks (TGNNs) have emerged as a capable tool for applications on graphs that evolve over time, like social networks or financial transaction graphs. Despite the black-box nature of these TGNNs, the explainability of TGNNs remains mostly unexplored. Particularly, counterfactual explanations, which establish what changes to the input graph result in an alternative model outcome, have not yet been explored to explain TGNNs.

To address explainability in TGNNs, this thesis introduces two novel counterfactual explanation methods, named **Gree**dy Explainer for Models on **Dy**namic Graphs using **C**ounter**f**actuals (GreeDyCF) and **Co**unterfactual Explainer for Models on **Dy**namic Graphs (CoDy). The explanation task is conceptualized as a search problem, seeking alterations to the input graph that change model outcomes. GreeDyCF performs a search that greedily traverses the search space, while CoDy leverages a search algorithm based on Monte Carlo tree search to find counterfactual explanations effectively. Extensive experiments show that CoDy and GreeDyCF are capable explainers that provide concise and expressive explanations. In particular, the experiments reveal the efficacy of CoDy, which outperforms other methods and achieves an up to 145% higher success rate in identifying counterfactual input alterations.

Contents

List of Abbreviations

CNN Convolutional Neural Network.

CoDy **Co**unterfactual Explainer for Models on **Dy**namic Graphs.

CTDG Continuous-Time Dynamic Graph.

DTDG Discrete-Time Dynamic Graph.

GNN Graph Neural Network.

GreeDyCF **Gree**dy Explainer for Models on **Dy**namic Graphs using **C**ounter**f**actuals.

GRU Gated Recurrent Unit.

LIWC Linguistic Inquiry and Word Count.

LSTM Long Short-Term Memory network.

MCTS Monte Carlo Tree Search.

RNN Recurrent Neural Network.

TGN Temporal Graph Network.

TGNN Temporal Graph Neural Network.

1 Introduction

Recent years have witnessed remarkable strides in artificial intelligence, with a particular surge in the prominence of deep neural networks [49]. Convolutional Neural Networks have revolutionized image-related tasks [57], and the advent of transformer-based language models [60, 12] is redefining how many tasks in natural language processing are performed. These advancements harness the intrinsic patterns within data structures, for example, the grid structure of images. Using these intrinsic patterns allows contemporary deep learning methods to effectively mitigate the challenges posed by an exponential increase in data volume and computational complexity as the number of dimensions in a dataset grows [11, 10, 9].

Research on deep neural networks is advancing to deploy and apply deep learning to data that is structured as graphs [66]. Deep learning models on graphs follow the success of models for image and text data in exploiting regularities in the structure of the underlying data [10]. Such models, referred to as Graph Neural Networks (GNNs), have demonstrated their prowess, for example, contributing to drug discovery through protein folding [27] or improving the precision of weather pattern forecasts [35].

While GNNs are predominantly applied to graphs that are static, meaning they do not change over time, many real-world domains are better described as dynamic graphs. Dynamic graphs are graphs that evolve over time, for instance, social networks, where connections form and dissolve over time [51, 55], road networks with traffic conditions that constantly change [76], or financial transaction networks, where transactions between entities can happen at any given moment. In dynamic graphs, nodes and edges can be added and deleted over time, and their features can evolve. Disregarding the temporal dimension for time-evolving data not only overlooks crucial temporal information for refining the model [69] but also risks the model mistakenly using future information to fulfill a task during training or validation, leading to faulty inference [69]. Tailoring models specifically for dynamic graphs has proven successful, outperforming their static counterparts in handling dynamic graph data [42, 43, 51, 59, 69, 55]. Such models are referred to as Temporal Graph Neural Networks (TGNNs). The dynamic graph data they operate on is usually represented as a sequence of timestamped events that each correspond to an atomic alteration of the graph, like the addition or removal of a new node or edge to the graph.

An example that illustrates the evolving nature of dynamic graphs is financial transaction data. In a financial transaction graph, people are represented as nodes, and transactions between people as timestamped edges between people. In such a graph, the inclusion of a temporal dimension allows answering questions like "Is a potential future transaction likely to happen or not?" with varying answers depending on the current time and state

of the graph. Such information can be used as decision support for spotting fraudulent transactions.

While deep models on dynamic graphs are getting increased attention [38], they remain black-boxes to their users [63]. This means that their inputs and outputs are known, but they do not provide any transparency or insights into why they produce a certain output. Yet, such explanations of the output are crucial in providing reliable and transparent predictions, particularly in critical areas like finance and healthcare [49]. Regulatory bodies are increasingly recognizing this necessity and have started to mandate explainability for tools that inform decisions in critical fields [45]. Existing explanation approaches primarily cater to GNNs operating on static graphs [73, 28]. Thus, these methods lack consideration for the temporal dimension of dynamic graphs. Without the ability to interpret temporal dependencies in the graph structure, these methods fall short of adequately explaining dynamic graph models [22, 67, 37]. Explanation methods for black-box models on dynamic graphs remain understudied [63, 38], with only a handful of publications addressing the intricacies of this topic [22, 67, 68, 37, 19].

Most existing explanation approaches require extensive access to the internals of the temporal models they explain. Only two prior works explain models that support dynamic graphs with a high temporal resolution. Furthermore, all existing methods provide factual explanations. Factual explanations detail the specific nodes, edges, and other features within the input graph that influence and contribute to a particular prediction [58]. However, they do not explore which changes to the input graph lead to different predictions. Counterfactual explanations, which are an increasingly popular alternative to factual explanations [49], address this shortcoming. Counterfactual explanations explain a model's prediction by showing how altering specific elements in the input graph, such as removing or adding nodes or edges, would result in a different prediction outcome [14]. This establishes causal relationships between the data and the model output and outlines decision boundaries [49]. Counterfactual explanations are particularly useful to users of black-box models because they provide actionable insights into the models' reasoning [39], they can be used to identify biases [49], and they can help in finding potential adversarial examples [39]. Despite the utility of counterfactual explanations, none of the existing explanation methods tailored for models on dynamic graphs has harnessed this potential until now.

To fill this gap that currently exists in research, this thesis employs counterfactual explanations to explain black-box models on dynamic graphs. It proposes two novel model-agnostic explanation approaches that provide counterfactual explanations. These approaches use a search-based approach to alter the input to a targeted model. For any prediction, the search aims to identify a subset of past events such that removing this subset from the input leads the targeted model to change its prediction. Such a subset is referred to as a counterfactual example. The first search approach is called **Gree**dy Ex-

plainer for Models on **Dy**namic Graphs using **C**ounter**f**actuals (GreeDyCF). It leverages a simple greedy approach to the search task and serves as a low-complexity explanation approach and as a baseline for counterfactual explanations for dynamic graph models. The second explanation approach is called **Co**unterfactual Explainer for Models on **Dy**namic Graphs (CoDy). It adapts search principles from Monte Carlo tree search to traverse the search space efficiently.

An extensive evaluation shows that CoDy is a capable counterfactual explanation method that outperforms GreeDyCF and a state-of-the-art factual explainer. Further, the evaluation suggests that guiding the proposed explainers is the exploration of the search space with temporal, spatio-temporal, and local gradient information about past events significantly improves their ability to find relevant counterfactual explanations.

In summary, the main contributions of this thesis are:

- **Problem Formulation:** This work formulates the problem of counterfactual explanations on dynamic graphs as a search problem. The properties of the search space are outlined, and the search space is structured in a way that facilitates finding explanations with minimal complexity.
- **Methods:** Two novel counterfactual explainers named GreeDyCF and CoDy are proposed to explain predictions of black-box models on dynamic graphs.
- **Experiments:** This thesis proposes a structured approach for conducting a comprehensive evaluation of factual and counterfactual explanation methods in the context of predictions on dynamic graphs. The evaluation shows that CoDy outperforms GreeDyCF and a state-of-the-art factual explanation method in a combined assessment of factual and counterfactual aspects of the explanations.

This thesis is structured as follows. As this work aims to explain black-box models on dynamic graphs, Section 2 outlines the foundations of static graphs and GNNs and how these are extended by a temporal dimension. Subsequently, Section 3 provides a taxonomic overview of how regular GNNs are explained with a focus on the differences between factual and counterfactual explanation methods. Following this, Section 4 details the current state of research into explanation methods for models on dynamic graphs. It establishes a substantial gap in research that is addressed in this thesis. In the subsequent section (Section 5), the counterfactual explanation problem for dynamic graphs is formally defined. This problem is addressed by the novel explanation methods GreeDyCF and CoDy, which are introduced in Section 6. An extensive evaluation is undertaken in Section 7 to establish the capabilities of the proposed explanation methods. Finally, Section 8 concludes the thesis, summarizing the thesis, discussing the limitations of this work, and outlining room for future research.

2 Deep Learning on Graphs

The foundation of this research lies at the intersection of graph theory, machine learning, and temporal data analysis. This chapter acts as a guide, leading the reader through the key concepts and methodologies that underpin this thesis. A core preliminary for explaining deep graph models on dynamic graphs lies in dynamic graphs themselves. Thus, chapter 2.1 establishes the definitions and fundamental differences between static and dynamic graphs (Section 2.1). After that, geometric deep learning is outlined (Section 2.2) as the basis for deep graph models, followed by an in-depth introduction to GNNs (Section 2.3). Building upon this foundation, the final sub-chapter (Section 2.4) covers the extension of GNNs into the temporal domain. It introduces GNN models designed for different representations of dynamic graphs. While Section 2.1 introduces the notation and vocabulary used throughout this thesis, Sections 2.2 - 2.4 introduce the concepts that underpin the explained models.

2.1 Graphs

On the most fundamental level, graphs are mathematical structures for modeling pairwise relationships between objects [17]. They find application in diverse domains like social and communication networks [11]. Graphs can be represented and denoted in a variety of ways. This thesis employs a notation based on the notation of Diestel [17], You et al. [71], and Souza et al. [55] to introduce and explore essential graph-related concepts.

A graph is characterized as a pair $G = (V, E)$, consisting of a set of vertices $V = \{v_1, ..., v_n\}$, also called nodes, and a multi-set of edges $E \subseteq V \times V$ [17, 71]. Each edge $e_i \in E$ is associated with two vertices, which are called its ends [17]. If a vertex $v_j \in V$ is the end to an edge $e_i \in E$, then v_j is designated as incident to e_i [17]. An edge $e_i \in E$ connecting a node $v_j \in V$ with itself is called a loop [17]. Another vertex $v_k \in V$ is adjacent, or a neighbor, to vertex v_j if they are connected by an edge [17].

Two vertices $v_0, v_n \in V$ can be linked by a walk $W(v_0, v_n)$, which is an alternating sequence of vertices and edges $W(v_0, v_n) = (v_0, e_0, v_1, ..., e_n, v_n)$ so that for all $j = 1, .., n$ the nodes v_{j-1} and v_j are the end points of edge e_j [17]. The length of a walk is the number of edges in the walk [17].

Furthermore, there exist different fundamental graph types. The graph types provide a richer vocabulary for modeling complex relationships and structures. The important graph types in the scope of this thesis are listed in the following.

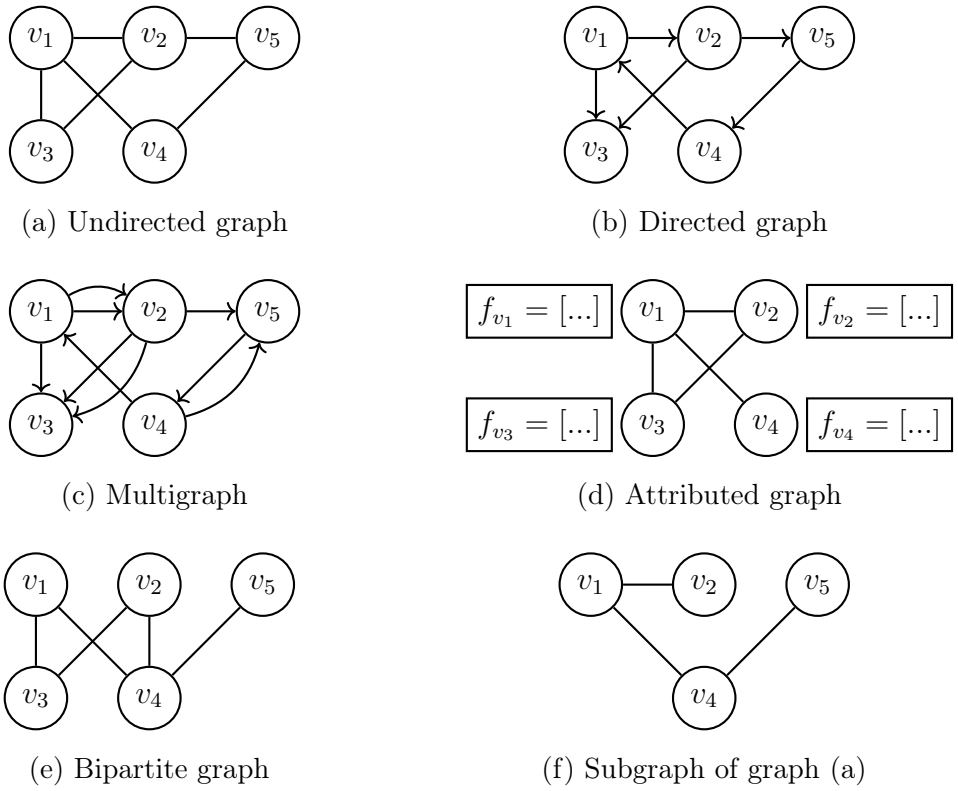
- In a **directed graph** each edge $e_i \in E$ connecting vertices $v_j \in V$ and $v_k \in V$ has a designated stating point and end point [17]. Therefore, each edge is represented as tuple $e_i = (v_j, v_k)$ with initial vertex v_j and terminal vertex v_k.
- In an **undirected graph**, edges have no direction, meaning that edge $e_i \in E$ connecting vertices $v_j \in V$ and $v_k \in V$ is represented as set $e = \{v_j, v_k\}$.
- A **multigraph** is a type of graph where multi-edges are permitted. This means that for any edge $e_i = (v_j, v_k) \in E$, there can exist multiple other edges $e_l = (v_j, v_k) \in E$ that share the same ends [29].
- An **attributed** graph associates properties with vertices and/or edges to represent their characteristics [29]. Node attributes are denoted as $A^{vertex} = \{a_{v_i}^{vertex} | v_i \in V\}$, and edge attributes as $A^{edge} = \{a_{e_i}^{edge} | e_i \in E\}$. The attributes are also referred to as features.
- A graph is called r**-partite** if the node set V can be partitioned into r classes so that all edges have their ends in different classes [17]. A 2-partite graph is generally referred to as **bipartite** graph [17].

Figure 1 shows examples of graphs with different characteristics. Several of these characteristics can coexist within a single graph. For example, the graph in Figure 1c is a directed multigraph.

Another critical aspect of graph theory is the concept of subgraphs. A graph $G' = (V', E')$ is called a subgraph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$ [17]. For example, Figure 1f portrays a subgraph derived from the graph in Figure 1a.

In addition to the structural definitions of graphs, there are further graph-related concepts that are important in the scope of this work. A fundamental concept is the local neighborhood $N_1(v_i) = \{v_j \in V | (v_i, v_j) \in E\}$ of a node $v_i \in V$, which consists of those nodes that are adjacent to v_i. The local neighborhood $N_1(v_i)$ is also referred to as the 1-hop-neighborhood of v_i because it exclusively contains nodes at a distance of 1 edge from v_i [10]. More broadly, the k-hop-neighborhood $N_k(v_i)$ of a node v_i encompasses nodes connected to v_i by walks with a length of at most k.

The graphs presented up to this point are commonly referred to as static graphs [29, 51, 71] because they capture relationships that remain constant. However, the real world rarely remains static; it continuously evolves, adapts, and undergoes change. To address the need for modeling these dynamic transformations, the concepts of static graphs are extended by a temporal dimension to encompass so-called dynamic graphs. Dynamic graphs introduce the element of time, allowing the capture of how relationships evolve and persist over time. In doing so, they provide a richer, more nuanced understanding of interconnected systems,



(a) Undirected graph

(b) Directed graph

(c) Multigraph

(d) Attributed graph

(e) Bipartite graph

(f) Subgraph of graph (a)

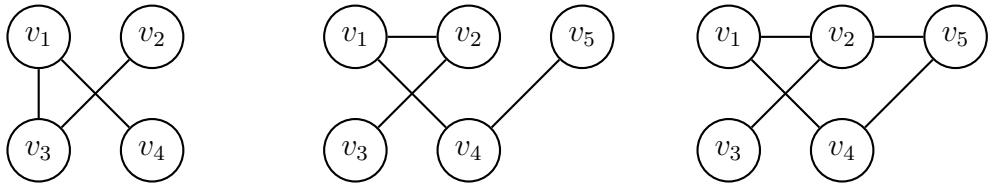
Figure 1: Examples of different graphs. Graphs (a) to (e) exhibit different characteristics, and graph (f) is a subgraph of the undirected graph (a).

closely mirroring real-world processes [71, 69, 59]. Examples of real-world processes that are best modeled as dynamic graphs are social networks, in which connections between individuals constantly form and end, and road networks with traffic information that continuously changes.

Two main approaches for modeling dynamic graphs have been proposed: First, modeling evolving graphs as a series of different static snapshots of a graph, so-called Discrete-Time Dynamic Graphs (DTDGs) [22, 68]. Second, modeling dynamic graphs as a timed list of interaction events, so-called Continuous-Time Dynamic Graphs (CTDGs) [51, 59]. In the following sections, these approaches are explored to establish the foundation of this work.

2.1.1 Discrete-Time Dynamic Graphs

Discrete-Time Dynamic Graphs (DTDGs) represent dynamic graphs as a series of static graph snapshots captured at different points in time [51]. These snapshots are usually taken with a constant temporal distance [55]. The changes between the snapshots represent the dynamic evolution of the graph. Formally, a dynamic graph in discrete-time representation is defined as $\mathcal{G} = \{G_t | t = 1, ..., T\}$, where T denotes the number of timestamps for which a snapshot is recorded [71]. Each snapshot represents a static graph $G_t = (V_t, E_t)$ with a set of nodes $V_t = \{v \in V | \tau_v = t\}$ and a set of edges $E_t = \{e \in E | \tau_e = t\}$ at a specific time $t \in \{1, ..., T\}$. Each vertex v and edge e carries a timestamp τ_v and τ_e respectively [71]. V and E represent the sets of nodes and edges aggregated across all timestamps. Furthermore, in an attributed DTDG each snapshot is associated with node features $A_t^{vertex} = \{a_{v,t}^{vertex} | v \in V_t\}$ and edge features $A_t^{edge} = \{a_{e,t}^{edge} | e \in E_t\}$. Compared to its static counterpart, the temporal 1-hop-neighborhood $N_1(v_i, t) = \{v_j \in V_t | (v_i, v_j) \in E_t\}$ of a node $v_i \in V$ additionally depends on the time t. Using the DTDG representation, node and edge additions and deletions, as well as attribute updates to node and edge features can be modeled. Figure 2 shows an example of a DTDG.



(a) Snapshot G_1 (b) Snapshot G_2 (c) Snapshot G_3

Figure 2: Example of a DTDG $\mathcal{G} = \{G_1, G_2, G_3\}$ at different snapshots. Between G_1 and G_2 the edge $\{v_1, v_3\}$ is removed, node v_5 is added and edges $\{v_1, v_2\}$ and $\{v_4, v_5\}$ are added. From G_2 to G_3 another edge $\{v_2, v_5\}$ is added.

While DTDGs offer simplicity and ease of processing through their snapshot-based representation [29], they sacrifice detailed structural information by design, leading to a coarse depiction of temporal data [59, 29]. Consequently, they may struggle to capture small and irregular temporal changes [59, 55]. For example, this representation may miss the addition and prompt removal of an edge. Moreover, selecting an appropriate granularity for taking snapshots often results in suboptimal representations [59]. Reducing the time interval between snapshots can mitigate these limitations. Still, it introduces complexity by including the entire static graph in each snapshot, leading to parameter redundancy due to shared nodes and edges between snapshots. These shortcomings are addressed by Continuous-Time Dynamic Graphs.

2.1.2 Continuous-Time Dynamic Graphs

Continuous-Time Dynamic Graphs (CTDGs) offer a high temporal granularity [59] and are efficiently represented as sequence of timestamped events $\mathcal{G} = \{\varepsilon_1, \varepsilon_2, ...\}$ [51]. Following Rossi et al. [51], each event ε_i is associated with a timestamp t_i with $t_1 \leq t_2 \leq ....$ There are node-wise events and interaction events. A node-wise event expresses the addition, removal, or attribute update of a node. Node-wise events are defined as a tuple:

$$\varepsilon_i = (v_j, a^{vertex}_{v_j,t_i}, type_i, t_i) \tag{2.1}$$

The tuple contains a timestamp t_i that denotes the time at which the event happens. Further, it includes an event type $type_i \in \{\text{add}, \text{delete}, \text{update}\}$ that describes what the event represents, e.g., either a node addition, a node deletion, or a feature update for a node. Additionally, it includes the node v_j that is added, deleted, or updated, as well as its node features $a^{vertex}_{v_j,t_i}$ at that point in time. As example, the event $\varepsilon_1 = (v_1, (1, 1, 1), \text{add}, t_1)$ describes the addition of node v_1 with attributes $(1, 1, 1)$ to the graph at time t_1.

An interaction event represents the addition, removal, or attribute update of an edge. Interaction events are defined as a tuple:

$$\varepsilon_i = (e_j, a^{edge}_{e_j,t_i}, type_i, t_i) \tag{2.2}$$

Similar to node-wise events, interaction events are also associated with a timestamp t_i that denotes the time of the occurrence of the interaction event and an event type $type_i \in \{\text{add}, \text{delete}, \text{update}\}$ that defines what type of an event ε_i is. Interaction events include an edge e_j that is part of the interaction, as well as the attributes of this edge $a^{edge}_{e_j,t_i}$. As an example, the event $\varepsilon_5 = (e_1, (1, 1, 1), \text{add}, t_5)$ with $e_1 = (v_1, v_2)$ describes the addition of an edge e_1 between nodes v_1 and v_2 with attributes $(1, 1, 1)$ to the dynamic graph at time t_5.

Depending on whether an event ε_i is a node-wise event or an interaction event, it has one or two so-called involved nodes. The involved node in a node-wise event is the node that the event adds, deletes, or updates. The involved nodes in an interaction event are those two nodes that are connected by the edge that is added, deleted, or updated by the event.

The node set V and the edge set E include all nodes and edges that are involved in any event in $\mathcal{G}$. A snapshot $G_{t_i} = (V_{t_i}, E_{t_i})$ of a CTDG $\mathcal{G}$ is a representation of $\mathcal{G}$ at time t_i as a static graph. V_{t_i} is the node set and E_{t_i} the edge set in the snapshot G_{t_i}. The snapshot G_{t_i} at time t_i is constructed from all events $\mathcal{G}(t_i) = \{\varepsilon_1, \varepsilon_2, ..., \varepsilon_i\}$ that occurred before or at timestamp t_i.

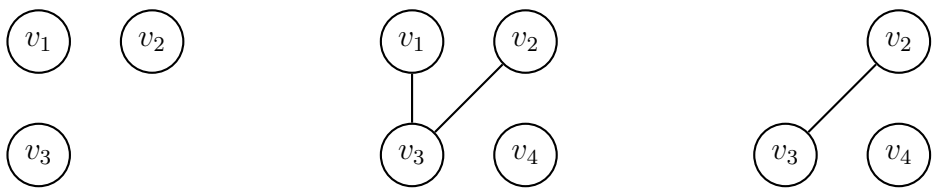
As in DTDGs, the temporal 1-hop-neighborhood of a node $v_i \in V$ at time t_l is defined as $N_1(v_i, t_l) = \{v_j \in V_{t_l} | (v_i, v_j) \in E_{t_l}\}$. The temporal-$k$-hop-neighborhood consists of those nodes $v_j \in V_{t_l}$ that are connected to v_i with a walk in E_{t_l} that has a length of at most k.

For CTDGs, the concept of temporal neighborhoods is extended from nodes to events. The temporal k-hop-neighborhood $N_k(\varepsilon_i)$ of an event ε_i consists of those past events that involve nodes that are in the temporal k-hop-neighborhood of the nodes involved in the event ε_i. This means an event $\varepsilon_j \in (\mathcal{G}(t_i) \setminus \varepsilon_i)$ is in the temporal k-hop-neighborhood of ε_i if any node involved in event ε_i is in the temporal k-hop-neighborhood of any node involved in the event ε_j.

Table 1 and Figure 3 illustrate an example of a CTDG $\mathcal{G}$, showing the events in Table 1 and the evolving graph in Figure 3. Table 1 demonstrates that this representation provides a highly detailed yet compact representation of the dynamic graph evolution. This is a significant advantage over the representation as DTDG [59]. The drawback of CTDGs is that an evaluation of the graph's state at any given timestamp necessitates the examination of all past events and the assembly of the graph at that specific moment. In comparison, the snapshots that comprise a DTDG already have a graph structure.

Event	Description
$\varepsilon_1 = \{v_1, a^{vertex}_{v_1,t_1}, \text{add}, t_1\}$	Add vertex v_1
$\varepsilon_2 = \{v_2, a^{vertex}_{v_2,t_2}, \text{add}, t_2\}$	Add vertex v_2
$\varepsilon_3 = \{v_3, a^{vertex}_{v_3,t_3}, \text{add}, t_3\}$	Add vertex v_3
$\varepsilon_4 = \{e_1 = \{v_1, v_3\}, a^{edge}_{e_1,t_4}, \text{add}, t_4\}$	Add edge e_1 between vertices v_1 and v_3
$\varepsilon_5 = \{e_2 = \{v_2, v_3\}, a^{edge}_{e_2,t_5}, \text{add}, t_5\}$	Add edge e_2 between vertices v_3 and v_3
$\varepsilon_6 = \{v_4, a^{vertex}_{v_4,t_6}, \text{add}, t_6\}$	Add vertex v_4
$\varepsilon_7 = \{e_1 = \{v_1, v_3\}, a^{edge}_{e_1,t_7}, \text{delete}, t_7\}$	Remove edge e_i between vertices v_1 and v_3
$\varepsilon_8 = \{v_1, a^{vertex}_{v_1,t_8}, \text{delete}, t_8\}$	Remove vertex v_1
$\vdots$	$\vdots$

Table 1: Sequence of events comprising an example graph $\mathcal{G}$.



(a) G_{t_3} (b) G_{t_6} (c) G_{t_8}

Figure 3: Different example snapshots of the graph $\mathcal{G}$ from Table 1 at timestamps t_3, t_6 and t_8.

2.2 Geometric Deep Learning

A major hurdle for applying deep learning methods to graphs lies in their underlying non-euclidean structure [66]. This means that graph data cannot be mapped to an Euclidean space $\mathbb{R}^n$ (for some dimension n). Geometric deep learning extends deep neural networks to non-Euclidean domains, providing a framework for working with graph data [11, 10]. The core principle in geometric deep learning is to leverage inherent symmetries in data to improve the performance of deep learning approaches [10]. These inherent symmetries are used as inductive biases for deep models, which means that the models are designed in such a way that they can take advantage of the symmetries [10]. Symmetries are transformations of the input data that leave the underlying structure and relationships in the data unchanged [10]. These transformations can include translations, rotations, reflections, and other geometric operations [10]. When the output of a function remains unchanged for symmetric data instances, it is referred to as invariant to that transformation [11]. For instance, an image classification model that is invariant to rotations recognizes an object in an image regardless of its orientation.

The key structural property that provides a symmetry in graphs is that nodes in a graph are provided without a fixed order [10]. This means that the output of functions on graphs should be independent of the order of the nodes [10]. A function that satisfies this condition is called a permutation invariant function [10]. An example of a permutation invariant function on a set of numbers is the arithmetic mean function, which provides the same output regardless of the order of the numbers in the set.

Deep learning methods that apply permutation invariant functions to graph data are referred to as Graph Neural Networks (GNNs) [10]. The following section covers the specifics of how these models translate the findings of geometric deep learning to actual deep neural models.

2.3 Graph Neural Networks

Graph Neural Network (GNN) are potent tools for various learning tasks on relational and interaction data. They find successful applications in many domains, including drug discovery [16], weather forecasting [34], and reasoning on knowledge graphs [23].

The applications of GNNs are diverse, but their tasks are often similar. The common tasks are node, edge, and graph classification, node regression, and link prediction [66, 77].

- **Node, Edge, and Graph classification** are the tasks of learning a function that maps each node, edge, or graph to their corresponding label [30].
- **Node regression** involves predicting a continuous numerical value associated with a node [66].
- **Link prediction** is the task of predicting the presence or absence of edges between pairs of nodes [36].

To accomplish these tasks, GNNs typically learn representations for elements of graphs, like nodes and edges [77]. Learning these representations allows the models to understand and use the structural information present in graph-structured data. The following sections lay out a generalized view of GNNs and present GNNs as an encoder-decoder pair.

2.3.1 Encoder-Decoder Framework

Different existing approaches to GNNs use diverse notations and conceptualizations to describe their methodologies. This thesis follows [21] and [29] in describing GNNs as an encoder-decoder pair. In the encoder-decoder framework, the so-called encoder component produces representations for graph elements, referred to as embeddings. The decoder is then tasked with making predictions based on these embeddings. Depending on the graph and task, nodes, edges, relation types, subgraphs, and/or entire graphs are embedded [8]. In many cases, embeddings are only calculated for nodes. Thus, in the following, only node embeddings are discussed. Within the encoder-decoder framework, node embeddings are formally defined as:

Definition 1. An **Embedding** is a numerical representation h_v for a node $v \in V$ in a graph. This representation commonly consists of one or more numerical scalars, vectors, or matrices [29]. For a given graph $G = (V, E)$, node embeddings are the result of the embedding function $emb : (v, G) \mapsto h_v \ \forall v \in V$.

Embeddings should capture and represent the structural and semantic information of nodes, facilitating information propagation, integrating local and global features, and enabling effective graph reasoning in downstream tasks [20]. The embedding function *emb* should thus provide similar embeddings for nodes that have structurally and semantically similar neighborhoods and similar features [20, 10]. Often, embeddings take the form of a single vector $h_v \in \mathbf{R}^d$ that has a significantly lower dimensionality than the number of nodes in the graph $d << |V|$ [20].

Within the encoder-decoder framework, the encoder refers to the node embedding function *emb*. The encoder typically builds on top of geometric deep learning principles and includes trainable parameters, such as neural network layers or modules. There are many different architectural variants for encoders, which are discussed in more detail in Section 2.3.2. In contrast, the decoder component is formally defined as:

Definition 2. Given an embedding h_v produced by the embedding function *emb*, the decoder makes predictions y for a particular task.

$$y = Decoder(h_v) \tag{2.3}$$

Like the encoders, decoders often incorporate trainable parameters [21]. The decoder architecture depends on the desired output format and the task it handles [21]. In link prediction, for example, the decoder may predict the likelihood of the existence of an edge, given the embeddings of the nodes in the predicted edge.

2.3.2 Graph Neural Network Architectures

Fundamental differences in GNNs architectures arise in the encoder. While the exact realization of the encoder can be diverse, most approaches build upon the same foundational structure [10]. GNNs are structured in so-called GNN layers [10, 66, 77]. Each node $v_i \in V$ in a graph is associated with a representation $h^l_{v_i}$ that is updated by each of the layers in the GNN [10]. The representation usually comes in the form of a numeric vector. A GNN layer is a computational step that updates the representation of each node in a graph by aggregating information from neighboring nodes [66]. GNN layers operate on the graph structure, and the node features A^{vertex} of a graph G [10]. In each layer, a shared permutation invariant function σ is applied to local neighborhoods $N_1(v_i)$ of each node $v_i \in V$ [10]. The output of this permutation invariant function for a node $v_i \in V$ in the l-th layer of a GNN is an updated representation $h^l_{v_i}$ of v_i. Consecutive GNN layers use the updated node representations produced by their preceding layers as input to the permutation invariant function σ. In doing so, they aggregate information from outside the local neighborhoods into the representations of each node. The first layer uses the raw node features $h^0_{v_i} = a^{vertex}_{v_i}, \forall v_i \in V$ as input [21]. If the nodes have no features, the

initial representation is commonly initialized with random values [66]. The updated node representation that is output by the last layer of a GNN is the embedding h_{v_i} of a node. On a generalized level, GNNs obtain updated node representations in the l-th layer as:

$$h_{v_i}^{l} = \sigma(h_{v_i}^{l-1}, \bigoplus_{v_j \in N_1(v_i)} \psi(h_{v_j}^{l-1}, \cdots)) \tag{2.4}$$

Here, permutation invariance is achieved by aggregating features of the local neighborhood $N_1(v_i)$ of node $v_i \in V$ from the previous layer with a permutation invariant function $\oplus$. For example, this can be the sum, mean, or maximum function [10]. The function ψ is dependent on the exact GNN implementation and it represents the main differentiating factor of GNN architectures. ψ is applied to each node v_j in the local neighborhood of v_i. It takes the updated features $h_{v_j}^{l-1}$ from the previous GNN layer as input alongside other implementation-specific features.

Following Bronstein et al. [10], most GNN architectures are derived from one of three types of GNN layers. These layer types are mainly differentiated by how the permutation invariant function σ transforms the neighborhood features through different realizations of ψ.

- **Convolutional GNNs:** In this architecture, information from the local neighborhood is directly aggregated with fixed weights c_{v_i,v_j} that represent the importance of node v_j to the representation of a node v_i [10, 66].

$$h_{v_i}^{l} = \sigma(h_{v_i}^{l-1}, \bigoplus_{v_j \in N_1(v_i)} c_{v_i,v_j} \varphi(h_{v_j}^{l-1})) \tag{2.5}$$

- **Attention-based GNNs:** By assigning different weights to neighboring elements, attention-based operators can alleviate noise and enhance the quality of results [77]. Weights result from a learnable self-attention mechanism a [10].

$$h_{v_i}^{l} = \sigma(h_{v_i}^{l-1}, \bigoplus_{v_j \in N_1(v_i)} a(h_{v_i}^{l-1}, h_{v_j}^{l-1}) \varphi(h_{v_j}^{l-1})) \tag{2.6}$$

- **Message-passing-based GNNs:** This architecture treats graph layers as message-passing process in which nodes directly pass information to their neighbors [66]. A learnable message function φ passes information between neighboring nodes [10].

$$h_{v_i}^{l} = \sigma(h_{v_i}^{l-1}, \bigoplus_{v_j \in N_1(v_i)} \varphi(h_{v_i}^{l-1}, h_{v_j}^{l-1})) \tag{2.7}$$

σ and ψ usually are learnable functions [10]. The node embeddings are typically obtained as the updated node features from the final layer of a GNN [21]. These embeddings are then used as input to task-specific decoders [21]. The exact implementations for these decoders vary and depend on the task performed by the GNN.

2.4 Temporal Graph Neural Networks

Compared with static GNNs, Temporal Graph Neural Networks (TGNNs) introduce representations for the temporal dimension of evolving data. Existing TGNN approaches are differentiated by the dynamic graph representation they target. Some approaches target DTDGs [52, 44, 71], while others target CTDGs [51, 55, 42]. Many approaches combine GNN layers developed for static GNNs with Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs) to introduce a temporal dimension [66]. In general, TGNN methods learn time-dependent node-embeddings [38]. They are structured into layers, similar to static GNNs. The output of the last layer of a TGNN for a node v_i at time t_j is the time-dependent node-embeddings $h_{v_i}(t_j)$. The following sections generalize common TGNN approaches for both types of temporal graph representations with a unified notation.

2.4.1 Temporal Graph Neural Networks for Discrete-Time Dynamic Graphs

TGNNs for DTDGs combine an approach for handling snapshots of the entire graph at each time point with a mechanism for learning temporal dependencies across time steps [38]. The most common approach is to evolve embeddings produced by a static GNN over time [38]. Hence, a permutation invariant function σ, which is adapted from the permutation invariant function in static GNNs, is employed in this context. Formally, this function is adjusted to calculate updated features of node $v_i \in V$ at a given time $t \in \{1, ..., T\}$ as follows:

$$\sigma(h_{v_i}^{l-1}(t), \bigoplus_{v_j \in N_1(v_i,t)} \psi(h_{v_j}^{l-1}(t), \cdots)) \tag{2.8}$$

ψ is an implementation-specific function that may take additional features as input. As in static GNNs, different architectures can be realized through different implementations of ψ. For each timestep t, σ functions exactly the same way as its counterpart in static GNNs, using the snapshot G_t of the dynamic graph at time t as a basis.

σ is combined with a temporal update function ϕ to obtain the updated node features [71].

$$h_{v_i}^{l}(t) = \phi(h_{v_i}^{l}(t-1), \sigma(h_{v_i}^{l-1}(t), \bigoplus_{v_j \in N_1(v_i,t)} \psi(h_{v_j}^{l-1}(t), \cdots))) \tag{2.9}$$

The temporal update function ϕ encodes the temporal dynamics by including the node representation of the last timestep $h_{v_i}^{l}(t-1)$ [38]. The update function might take additional arguments, depending on the implementation. There exist various implementations of the permutation invariant function σ and the temporal update function ϕ [38]. These include using self-attention [52], or RNNs like a Gated Recurrent Unit (GRU) [71].

2.4.2 Temporal Graph Neural Networks for Continuous-Time Dynamic Graphs

TGNN methods for CTDGs effectively handle sequences of events by integrating methods that refresh a node's representation whenever an event related to that node takes place [38]. They represent an extension of static GNNs because they merge and aggregate node representations within temporal neighborhoods [38].

While the exact approaches vary significantly [38], a relatively generic framework is Temporal Graph Network (TGN) [51]. Since many popular approaches are specific instances of TGN [59, 51] or extend upon it [55], the following paragraphs describe TGNNs for CTDGs from the viewpoint of the TGN framework.

TGNNs for CTDGs introduce two novel concepts: Event-specific messages and node-specific state. Each node $v_i \in V$ is associated with a time-dependent state vector $s_{v_i}(t)$ that represents all its past interactions [38]. Whenever an event ε_k that involves node $v_i \in V$ occurs, a message $m_{v_i}(\varepsilon_k)$ is generated to update the node's state $s_{v_i}(t_k)$. For an interaction event ε_k between source node $v_i \in V$ and target node $v_j \in V$, it is possible to compute two messages:

$$m_{v_i}(\varepsilon_k) = msg_s(s_{v_i}(t_k^-), s_{v_j}(t_k^-), \varepsilon_k), \quad m_{v_j}(\varepsilon_k) = msg_t(s_{v_j}(t_k^-), s_{v_i}(t_k^-), \varepsilon_k) \tag{2.10}$$

t_k^- refers to the time just before t_k, meaning that $s_{v_i}(t_k^-)$ is the state of node v_i just before the event ε_k occurs. This formulation distinguishes between source and target nodes to represent directed graphs. Thus, separate message functions exist for updates to the source node state msg_s and the target node state msg_t. To update the node state in undirected graphs, both message functions are used to update each node with both the source node message msg_s and the target node message msg_t. In the case that the encountered event ε_k is a node-wise event on node $v_i \in V$, a single message is computed:

$$m_{v_i}(\varepsilon_k) = msg_n(s_{v_i}(t_k^-), \varepsilon_k) \tag{2.11}$$

The message functions msg_s, msg_t, msg_n can be learnable. A message $m_{v_i}(\varepsilon_k)$ updates the state $s_{v_i}(t_k)$ of node v_i trough a learnable state function *state*:

$$s_{v_i}(t_k) = state(m_{v_i}(\varepsilon_k), s_{v_i}(t_k^-)) \tag{2.12}$$

The *state* function is a RNN, often in the form of a Long Short-Term Memory network (LSTM) or GRU [38]. The state is used to compute node embeddings in the TGNN layers. These layers resemble their counterparts from static GNNs. On a generalized level, updated node features at time t are calculated as:

$$h_{v_i}^l(t) = \sigma(h_{v_i}^{l-1}(t), \bigoplus_{v_j \in N_1(v_i,t)} \psi(s_{v_j}(t), \cdots)) \tag{2.13}$$

Where σ and ψ can be learnable [51]. ψ takes the state of neighboring nodes as input, together with other features, depending on the specific implementation. Like in static GNNs, the actual mechanisms used in the permutation invariant function vary. Popular approaches include using RNNs [42] and a self-attention mechanism [51].

To improve scalability, modern TGNNs for CTDGs employ strategies like only updating the state periodically with an aggregated version of the messages [51]. There also exist approaches that do not use a separate state component. Instead, they embed nodes directly by combining node features, graph topology, and time embeddings [38, 69], similar to static GNNs.

3 Explaining deep graph models

Alongside the rise in adaptation of artificial intelligence and deep learning methods over recent years, the idea of explainable artificial intelligence has been gaining increasing attention [2]. The field of explainable artificial intelligence focuses on providing transparency over the mechanisms that underlie artificial models [7]. Many machine learning models behave like black-boxes, meaning that the internal mechanisms that lead to their predictions remain opaque to their users [49]. Explainable artificial intelligence differentiates *interpretability* from *explainability*. Interpretability refers to the degree to which the internal workings and decision-making processes of an artificial intelligence model can be understood by humans by design [7]. In comparison, explainability refers to the ability of an artificial intelligence system to provide clear, coherent, and accessible explanations for its decisions or predictions, offering insights into the rationale behind specific outcomes in a human-understandable manner [7]. While interpretable GNN models utilize mechanisms that are easy to understand on their own [49], black-box GNN models have shown better performance on high-dimensional input data than interpretable models [49]. Thus, proper explanation methods are required to make black-box models explainable.

Even though most works on static GNNs have only emerged in the past 10 years [66], the black-box nature of the majority of these GNNs has given rise to a plethora of explanation methods. The explanation approaches that have been proposed vary in many ways, like the type of explanation provided, the mechanism by which they explain produce explanations, or if and how explainers are trained [28]. Figure 4 provides a classification of these varied approaches. On a conceptual level, model-level and instance-level approaches are differentiated [73]. Model-level methods explain GNNs as a whole, irrespective of any particular input example [73]. Such explanations aim to uncover a high-level understanding of the inner workings of the explained GNN. In comparison, instance-level methods explain the output of a model for a specific input instance [73]. These explanations are tailored to individual instances, offering a fine-grained understanding of the reasoning behind each prediction.

Model-level explainers like XGNN [72] map the behavior of a GNN to patterns in the input graphs [73]. These explainers use a so-called generation approach, which explains GNNs by training a model to generate graphs that maximize a particular prediction of the GNN [73, 72]. The thereby generated graph patterns are then used as an explanation [72]. While these approaches can provide high-level explanations for the general model behavior, they fail to explain the specifics of why certain predictions are made for given inputs and are thus not discussed any further.

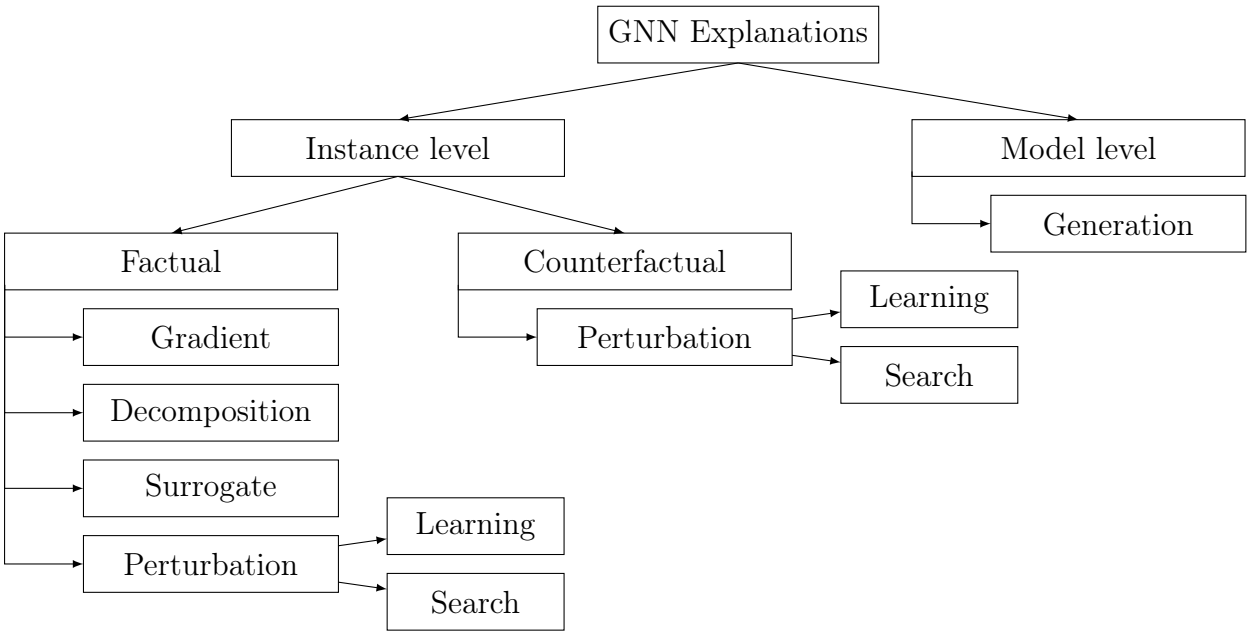


Figure 4: Taxonomy of GNN explanation methods. Explanation approaches are categorized into instance-level and model-level methods. The category of instance-level methods is further subdivided into factual and counterfactual approaches. Taxonomy adapted from [73], [49], and [28].

Instance-level approaches are subdivided into those that provide a factual and those that provide a counterfactual explanation. On a fundamental level, factual reasoning inquires, "With A already having occurred, will B occur?" [50, 58]. Hence, a factual explanation states that a prediction for B was made because A occurred. In contrast, counterfactual reasoning poses the question, "If A did not occur, would B still happen?" [50, 58]. A counterfactual explanation is thus interested in the information A, whose non-existence would entail a different prediction for the occurrence of B. The following sections take a closer look at explanation methods using either type of reasoning.

3.1 Factual Explanation Approaches

Factual explanations for GNNs answer the question: "Given the input A, will the GNN predict B?". These explanations seek the input information with the maximum influence on the GNN's prediction [28]. They can take the form of a subgraph $G' \subseteq G$ of the original input G [28] that is *sufficient* to produce the original prediction with the targeted GNN $f(\cdot)$ [58]. For a sufficient factual explanation, the following equation holds:

$$f(G) = f(G') \tag{3.1}$$

As depicted in Figure 4, Factual explainers for GNNs are categorized based on the primary mechanisms they employ to explain predictions, each offering different approaches and forms of explanations:

- **Gradient/Feature** based methods identify important features by analyzing how changes in input features affect the network's predictions [73]. These approaches either employ back-propagation to examine the gradients of the explained prediction concerning the input features or map hidden features through interpolation into the input space [73]. Examples of these methods are Class Activation Mapping (CAM)[47] and Sensitivity Analysis (SA) [6].

- **Decomposition** methods break down a GNN's prediction into contributions from individual nodes or edges by creating score decomposition rules through back-propagation [73]. These rules attribute the prediction scores to individual features in the input space [73, 6]. Notable decomposition approaches are Layer-wise Relevance Propagation (LRP) [6] and GNN-LRP [53].

- **Surrogate** methods create a separate, easy-to-interpret model to approximate the behavior of a GNN on data similar to the explained example [73]. Usually, neighboring data to the explained example is sampled and used to train easily analyzable models [24] like linear regression models [18] or Bayesian networks [62]. Relevant publications that leverage surrogate models for explanations are Probabilistic Graphical Model Explainer (PGM-Explainer) [62] and Graph Local Interpretable Model Explanations (GraphLIME) [24].

- **Perturbation**-based methods analyze how the output of a GNN model changes when the input is perturbed [73]. Input perturbations refer to alterations to the input graphs [73]. Perturbation-based methods usually create feature masks that highlight those input features that are important for the prediction and exclude the unimportant features [73, 25]. Perturbation-based methods operate with either a learning-based or a search-based perturbation approach [67]. Popular learning-based perturbation approaches are GNNExplainer [70] and PGExplainer [40]. GNNExplainer [70] uses a trainable model to generate masks for the input features, which requires training for each explained prediction separately. In contrast, PGExplainer [40] uses a trainable masking model that is only trained once for a graph and can then be applied to explain any prediction on that graph. In terms of search-based perturbation approaches, SubgraphX [74] is an influential method that uses Monte Carlo tree search to find perturbations that explain a prediction.

While gradient/feature and decomposition approaches require direct access to the internal model parameters or embeddings by design, some surrogate and perturbation-based methods can operate without such access [28]. Needing direct access is a disadvantage since it requires adaptations in the explainer before these methods can be applied to new models. It also prohibits the application of the explainers to models that do not expose their inner workings.

As mentioned description of the explanation mechanisms, perturbation-based methods use either a learning-based or a search-based perturbation approach [67]. Both of these approaches usually generate a perturbed subgraph of the original input graph, which serves as an explanation. Learning-based approaches identify the most critical features of the input graph by feeding the node embeddings and/or edge embeddings produced by the explained GNN into a learnable model that extracts a perturbed subgraph [28, 67]. The GNN predictions on this subgraph are then compared to the original predictions with a distance function, providing a score that serves as training input for the subgraph extraction model [28]. Search-based perturbation approaches generate modified subgraphs from the original input using heuristic search algorithms together with a game-theoretical scoring function [67, 74]. A clear advantage of the search-based perturbation strategy is that model access is only needed on the level of predictions, not for the embeddings. This comes at the cost of longer inference times [67] since the search-based approach requires the separate exploration of various input perturbations for each explained instance [74].

3.2 Counterfactual Explanation Approaches

Counterfactual explanations for GNNs answer the question, "What changes in the input graph are necessary to change the prediction produced by a GNN model?". Counterfactual explanations aim to identify so-called counterfactual examples. A counterfactual example consists of the smallest possible alteration to the input information such that the model prediction changes [28]. These counterfactual examples consist of the input information that is *necessary* to the prediction of the explained model [58]. If the information contained in a counterfactual example is excluded from the input, the prediction of the explained model changes [58]. Counterfactual explanations for GNNs have risen in attention over the last years [41] for their ability to produce intuitive explanations [41], offer feedback for non-experts [49], foster trust in GNNs [49], and help in addressing model biases [49]. Further, they are useful for identifying potential adversarial examples that could be used to attack models [39].

Typically, counterfactual examples consist of a subgraph $\mathcal{X} \subseteq G$ which if removed from the input graph G will result in the explained GNN $f(\cdot)$ producing a different prediction than on the full original graph G [58]:

$$f(G) \neq f(G \setminus \mathcal{X}) \tag{3.2}$$

In some cases, counterfactual examples also include information added to the original graph G [1]. In such instances, the definition of counterfactual examples is extended. In this extended definition a counterfactual example $\mathcal{X} = \{\mathcal{X}^-, \mathcal{X}^+\}$ consist of a graph $\mathcal{X}^- \subseteq G$, containing information removed from G, and a graph $\mathcal{X}^+ = (V^+, E^+)$ with $v_i \in V^+ \implies v_i \notin V$ and $e_i \in E^+ \implies e_i \notin E$ that contains information added to the input graph. This combination must satisfy the following equation:

$$f(G) \neq f((G \cup \mathcal{X}^+) \setminus \mathcal{X}^-) \tag{3.3}$$

All counterfactual explanation methods leverage a perturbation-based approach. Similar to the perturbation approaches for factual explanations, a difference between search-based and learning-based approaches is made.

- **Search-based** approaches either explore alterations to the input graph dependent on a specific criterion [49], like a similarity measure between separate alterations of a graph [1], or they use a systematic policy for modifying the input graph until they attain a valid counterfactual [49]. Examples of this type of counterfactual explainers are Molecular Model Agnostic Counterfactual Explanations (MMACE) [65] and Oblivious Backward Search (OBS) [1].
- **Learning based** approaches examine how the GNN output changes to variations in the input graph [49]. They train a model to perturb input graphs in a way that the output of the targeted model changes [39]. To find counterfactual examples, the trained model is used to produce multiple perturbations of the input graph while checking if the perturbations constitute counterfactual examples [39]. Various approaches within this category differ in how they create and update the input graph mask and how the model is trained [49]. A popular representative of this category is CF-GNNExplainer [39].

There also exist methods that are not fully captured by this classification. For example, RCExplainer [5] combines a learning-based approach with a learned model of the decision logic of a GNN.

Counterfactual explainers and GNNs are usually decoupled from each other [49]. Hence, the majority of counterfactual explanation methods for GNNs offer the advantage of being model agnostic.

3.3 Comparison of Factual and Counterfactual Approaches

In comparison, while factual explanations shed light on why a GNN makes a particular prediction given the input, counterfactual explanations go a step further by revealing precisely what changes in the input are needed to alter that prediction. Because factual reasoning seeks sufficient information as an explanation for a prediction, it may include redundant information [58]. In contrast, as counterfactual reasoning only seeks necessary information as explanation, it may only extract a small subset of the information that is actually responsible for the prediction [58]. For this reason, some works integrate factual and counterfactual reasoning into a single explanation method, for example, CF^2 [58]. This has the advantage of addressing the necessity and sufficiency of explanations within the same framework [58].

On their own, counterfactual explanations provide a practical and intuitive way for users to grasp what influences the explained model's output in an actionable manner [39]. By offering concrete, actionable insights into the model's decision-making process [39], they empower individuals to understand, trust, and improve GNNs in real-world applications [49]. Thus, counterfactual explanations bridge the gap between the model's black-box behavior and human interpretability, making them a valuable tool for ensuring transparency and accountability.

4 Related Work

As outlined in Chapter 3, numerous explanation methods for GNNs on static graphs have been developed. However, these models fall short when applied to temporal graph models, as they fail to capture temporal dependencies in dynamic graphs [67, 22]. Additionally, they are unable to support the DTDG and CTDG representations of dynamic graphs. Particularly, the event-based CTDG representation is conceptually substantially different from the static representation of graphs as mere sets of nodes and edges. Yet, only a very limited number of past publications have explored explanation methods that take the dynamic nature of TGNNs into account. To develop and evaluate novel explanation methods for TGNNs, it is crucial to understand existing explanation methods in the context of dynamic graphs. Hence, this chapter discusses explanation approaches that target models on dynamic graphs introduced by prior works. An overview of explanation methods for TGNNs in related works is presented in Figure 4. The methods used in these explainers mainly adapt findings from explainers in the static setting to the dynamic context.

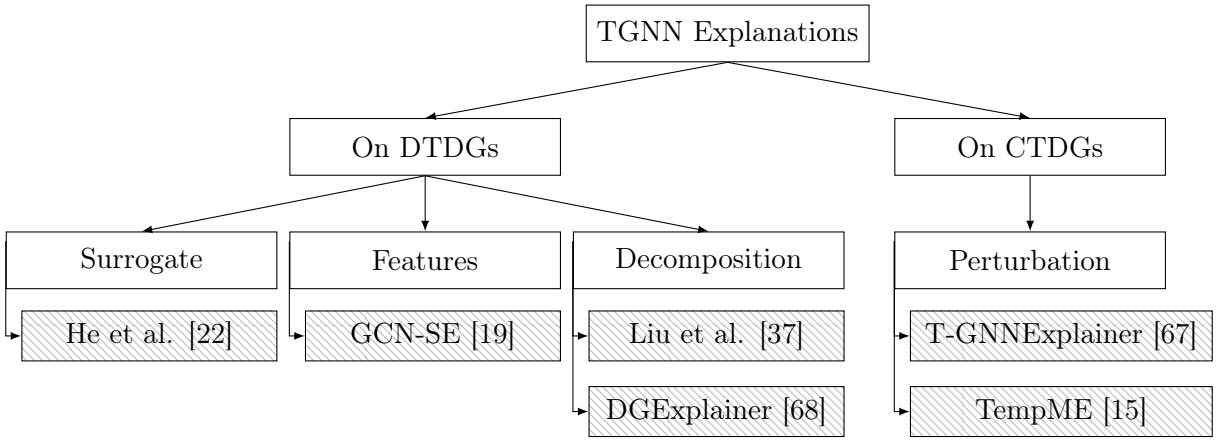


Figure 5: Overview of related work. Some works focus on explaining TGNNs operating on the DTDG representation, while only two other studies explore explanations for TGNNs operating on CTDGs. The methods are subdivided by their approach, following the taxonomy introduced for the static explanation approaches in Figure 4.

This chapter provides a structured overview of existing methods and how they compare to the approaches developed in this thesis. Distinguishing the approaches at the level of the underlying graph representation, the following sections outline related works for explanations in the DTDG (Section 4.1) and the CTDG setting (Section 4.2). Section 4.3 compares the existing approaches and outlines the gap that exists within contemporary research.

4.1 Explanations on Discrete Time Dynamic Graphs

Most prior work focuses on explanations for predictions on DTDGs. Explaining such models involves integrating feature importances over the different graph snapshots [22]. Existing approaches are based on surrogate models [22], temporal decomposition strategies [68, 37], and model features [19].

He et al. [22] develop a surrogate-based approach. Their explainer adapts PGM-Explainer [62], a surrogate method that explains static GNNs, to explain predictions on DTDGs. PGM-Explainer aims to construct an optimal Bayesian network that captures the most relevant graph components. It provides explanations in the form of conditional probabilities. The authors apply PGM-Explainer to each of the graph snapshots in a DTDG by treating the snapshots as static graphs. This results in a separate Bayesian network as an explanation for each snapshot. They subsequently aggregate these individual explanations using a pruning strategy to discover explanations that are dominant across the individual explanations.

In contrast, GCN-SE [19] takes a feature-based approach. GCN-SE itself is not an explainer but rather a graph convolutional network architecture for DTDGs with inbuilt explanation capabilities. It employs attention weights for the different graph snapshots. These attention weights are used to assess the importance of the different snapshots to the final prediction. Thus, the GCN-SE approach provides a degree of self-interpretability. However, the attention weights only offer insights into how important certain snapshots are and not into the importance of features within the snapshot, offering only coarse explanations.

Another category of explanation methods utilizes the decomposition approach. DGExplainer [68] decomposes the prediction of a TGNN by redistributing the prediction score to relevance scores of neurons in the previous neural network layer. This redistribution is repeated backward through the TGNN until the input neurons are reached. DGExplainer interprets the redistributed scores as importances for the input features. Liu et al. [37] also propose a decomposition-based explanation method. They use standard GNNs designed for static graphs to make predictions on the different snapshots. Then, they make the predictions understandable across the different snapshots using axiomatic attribution. The authors presume that the predicted distributions exist within a low-dimensional manifold. The approach demonstrates how these predictions change over time as smooth paths on this manifold. The main goal is to enhance interpretability by finding a path on the manifold that accurately represents how the predictions change over time.

4.2 Explanations on Continuous Time Dynamic Graphs

Explanations tailored for CTDGs is an area with very limited prior work. To the best of the author's knowledge, there exist only two explainers that focus on CTDGs, named T-GNNExplainer [67] and TempME [15]. They both use a perturbation approach to provide factual explanations in the form of a subset of past events crucial for the model's predictions.

The underlying perturbation approach of T-GNNExplainer [67] is search-based and leverages Monte Carlo Tree Search to find important events. To improve search performance, the authors introduce a navigator component, a multilayer perceptron trained to predict event importances concerning a given prediction. This navigator informs the exploration of the search space. Notably, T-GNNExplainer differs from many perturbation-based explanation methods in static settings in that it does not require users to specify the size of the explanation. Instead, it autonomously identifies an explanation of an appropriate size. T-GNNExplainer can be interpreted as an adaptation of SubgraphX [74], a search-based perturbation approach for static GNNs, to the dynamic setting. In contrast to this predecessor, T-GNNExplainer uses spatial and temporal constraints for its search space and adopts the navigator component for efficient search space explorations.

In contrast, TempME [15] is an approach that aims to identify crucial temporal motifs to elucidate the reasoning behind predictions. Temporal motifs are recurrent and statistically significant temporal substructures of the dynamic graph. TempME selects important temporal motifs sampled from a local neighborhood of the nodes involved in the explained prediction. For this selection, it uses a GNN pretrained on predicting the importance of motifs to the explained prediction. These importance ratings are used as basis to construct explanations that integrate the collective impact of the identified temporal motifs.

4.3 Comparison

While the presented related works provide valuable contributions to explaining predictions of TGNNs, they display different shortcomings and leave many possible explanation approaches unexplored. Table 2 summarizes the key characteristics of the presented approaches.

TGNNs operating on CTDGs are generally more versatile than their DTDG counterparts because DTDGs can be converted into CTDGs without loss of information, but not necessarily the other way around [55]. Despite this, explanations for TGNNs working on CTDGs remain understudied [15]. T-GNNExplainer [67] and TempME [15] are the only approaches yet to explore explanations in the context of CTDGs.

A major shortcoming of many existing explanation approaches is that they are not model

Method	Tasks	Approach	No Model Access	Model Agnostic	CTDG	CF explanations
He et al. [22]	NR	Surrogate	✓	✓	·	·
GCN-SE [19]	NC	Features	·	·	·	·
Liu et al. [37]	NC, GC, FLP	Decomposition	·	·	·	·
DGExplainer [68]	FLP, NR	Decomposition	·	·	·	·
T-GNNExplainer [67]	FLP	Perturbation	∼*	✓*	✓	·
TempME [15]	FLP	Perturbation	✓	✓	✓	·
GreeDyCF (this work)	FLP	Perturbation	✓	✓	✓	✓
CoDy (this work)	FLP	Perturbation	✓	✓	✓	✓

Table 2: Overview of related work compared to the explainer proposed in this thesis. The abbreviations for the tasks are Node Regression (NR), Node Classification (NC), Graph Classification (GC), and Future Link Prediction (FLP). *: The navigator component in T-GNNExplainer requires access to attention weights.

agnostic. A model agnostic explainer applies to any prediction model [49]; however, most of the related works require a specific class of model, for instance, an attention-based method. This means that such methods cannot be applied to black-box models.

Another factor limiting the applicability of most related explanation approaches is model access. Explanation methods can require access to the prediction model on different levels. Access on the level of gradients, embeddings [61], and predictions are differentiated. While approaches that require embedding-wise access only limit the scope of explainable models to those that produce such embeddings, approaches with gradient-wise access restrict the range of explainable models to those that employ accessible neural networks [49]. A reliance on model access to internal parameters can prohibit the application of an explainer to a novel model, or it might make rigorous adaptations necessary. Novel methods are often designed without any considerations for explainability or specific explainability methods [67], making a non-reliance on model access an asset for an explainer.

Adding to these limitations, none of the existing approaches makes use of counterfactual explanations even though counterfactual explanations have been shown to be highly effective explanations, as they overcome many problems in interpretability and accountability [64]. Additionally, counterfactual explanations have already been successfully applied to regular GNNs [58, 39], making their application in explaining models on dynamic graphs a logical next step.

This thesis advances the capabilities of explanation methods for black-box prediction models on dynamic graphs. It addresses the identified research gap by proposing **Gree**dy Explainer for Models on **Dy**namic Graphs using **C**ounter**f**actuals (GreeDyCF) and **Co**unterfactual

Explainer for Models on **Dy**namic Graphs (CoDy), two instance-level post-hoc explainers that provide counterfactual explanations for TGNNs operating on CTDGs. The proposed methods are model agnostic, treating the explained model as a black-box with model access only required on the level of predictions. Both approaches use a search-based perturbation strategy that seeks minimal counterfactual examples. In proposing GreeDyCF and CoDy, this thesis provides a valuable addition to the current state of explanation methods for dynamic graphs.

5 Problem Formulation

This chapter lays the foundation for exploring the explainability of prediction models on dynamic graphs by formulating the problem of finding counterfactual explanations on CTDGs. It introduces the task of future link prediction (Section 5.1). This task serves as the reference task for proposing the explanation method. Subsequently, counterfactual explanations are defined for future link prediction (Section 5.2). Section 5.3 presents the specific objectives pursued within the explanation approach. Finally, the overall goal of this thesis is summarized in Section 5.4.

5.1 Explanations for Future Link Prediction

In the context of future link prediction, a link prediction model $f : \{\mathcal{G}(t_i), \varepsilon_i\} \to \mathbb{R}$ is tasked with predicting whether a future edge addition event $\varepsilon_i = \{e_j, a^{edge}_{e_j,t_i}, \text{add}, t_i\}$, referred to as future link, will occur or not. The prediction is based on all events $\mathcal{G}(t_i) = \{\varepsilon_l | \varepsilon_l \in \mathcal{G}, t_l \leq t_i\}$ from the temporal graph $\mathcal{G}$ that happened before or at time t_i. Without loss of generality, it is assumed that f provides its prediction in the form of real-valued logits, defined as the quantile function of the standard logistical distribution. The logit values capture the model's confidence in the occurrence of the event. The model f can be any function that fits this description, for example, a TGNN trained on future link prediction. In this task, the model f is also referred to as link prediction function. The binary classification function $p : \mathbb{R} \to \{0, 1\}$ is applied to the output of the link prediction function f to get the final interpretation of the prediction:

$$p(f(\mathcal{G}(t_i), \varepsilon_i)) = \begin{cases} 1, & \text{if } f(\mathcal{G}(t_i), \varepsilon_i) \geq 0 \\ 0, & \text{else} \end{cases} \tag{5.1}$$

If the classification function outputs a value of 1, the prediction is classified as a forecast of the future link's occurrence. If it outputs 0, the prediction is classified as a forecast of the future link's non-occurrence.

An explainer is a function that provides an explanation for a model's prediction. Within the context of future link prediction, an explainer $ex(\cdot)$ is a function that takes the link prediction function f, the explained event ε_i, and the list of past events $\mathcal{G}(t_i)$ as input and outputs an explanation for the model's prediction.

While the explanation approach proposed in this thesis primarily focuses on the domain of future link prediction, the insights gained can be generalized to adapt to node and graph classification tasks with minor adjustments.

5.2 Counterfactual Examples

As discussed in Section 3, counterfactual explanations are a valuable tool for providing intuitive and actionable explanations regarding "why" a specific prediction was made. Thus, they are particularly well-suited for explaining future link predictions made by difficult-to-understand deep graph models like TGNNs.

A counterfactual example $\mathcal{X}_{\varepsilon_i}$ to a model's prediction on the occurrence of a future link ε_i consist of a subset of past events $\mathcal{X}_{\varepsilon_i} \subseteq \mathcal{G}(t_i)$. To be considered counterfactual, the subset has to be necessary to produce the original prediction, satisfying the following condition:

$$p(f(\mathcal{G}(t_i), \varepsilon_i)) \neq p(f((\mathcal{G}(t_i) \setminus \mathcal{X}_{\varepsilon_i}), \varepsilon_i)) \tag{5.2}$$

For any potential future link ε_i, there may exist multiple counterfactual examples or no counterfactual example at all. The set containing all existing counterfactual examples for a potential future link ε_i is denoted as Ψ_{ε_i}.

A counterfactual explainer for the future link prediction task ex is defined as a function that takes a link prediction function f, a temporal graph $\mathcal{G}(t_i)$, and a potential future link ε_i as input. It outputs a subset of the events from the original input graph as a counterfactual explanation.

$$ex : \{f, \mathcal{G}(t_i), \varepsilon_i\} \rightarrow \bigcup_{k=0}^{|(\mathcal{G}(t_i) \setminus \varepsilon_i)|} \binom{(\mathcal{G}(t_i) \setminus \varepsilon_i)}{k} \tag{5.3}$$

The definition constrains the output of the explainer to any combination of past events. This definition follows the notation in [56] for representing the set of all permutations of a particular size k.

5.3 Objectives of the Explainer

Explaining future link predictions using counterfactual examples entails two primary objectives:

1. **Maximize Discoveries**: The explainer's first objective is to maximize the proportion of cases in which it successfully identifies a counterfactual example. Every subset of past events could potentially constitute a counterfactual example. Given that the number of possible combinations of past events grows exponentially with the number of past events, it quickly becomes infeasible to explore all possible combinations. Thus, the explainer ex should seek to efficiently identify counterfactual examples, maximizing the instances in which the explanation is necessary to the explained prediction. This is achieved by maximizing the following expression:

$$\frac{1}{\mathcal{G}} \sum_{\varepsilon_i \in \mathcal{G}} \mathbb{1}(ex(f, \mathcal{G}(t_i), \varepsilon_i) \in \Psi_{\varepsilon_i}) \tag{5.4}$$

   The indicator function $\mathbb{1}(ex(f, \mathcal{G}(t_i), \varepsilon_i) \in \Psi_{\varepsilon_i})$ returns 1 if the explainer ex finds a counterfactual example for the given input and returns 0 otherwise.

2. **Minimize Complexity**: When multiple counterfactual examples exist, the explainer should select the most concise yet informative explanation. In line with Occam's razor principle, good explanations should be simple, comprising only the most relevant information [73, 58]. Hence, counterfactual examples should have minimal sizes, as measured by the number of events they contain:

$$\arg\min_{\mathcal{X}^j_{\varepsilon_i} \in \Psi_{\varepsilon_i}} |\mathcal{X}^j_{\varepsilon_i}| \tag{5.5}$$

   where the function $|\mathcal{X}^j_{\varepsilon_i}|$ returns the number of events that comprise $\mathcal{X}^j_{\varepsilon_i}$.

5.4 Problem Statement

In summary, this thesis proposes explaining predictions made by TGNNs in the context of future link prediction using counterfactual explanations. Given a CTDG $\mathcal{G} = \{\varepsilon_1, \varepsilon_2, ...\}$, and a trained temporal graph model f, an explainer ex provides explanations for why the model predicts either the occurrence or non-occurrence of a future link ε_i. The explanation consists of a subset of the past events $\mathcal{X}_{\varepsilon_i}$.

The primary objectives of a counterfactual explainer are two-fold: maximizing the number of cases in which it successfully identifies a counterfactual example while concurrently minimizing the complexity of the explanation.

6 Methodology

This chapter addresses the explanation problem defined in Section 5 using two explanation approaches. As the basis for these approaches, the task of explaining predictions of future link prediction models on CTDGs is framed as a search task. Section 6.1 defines the search space and explores the question of how to structure it in accordance with the objectives defined for explainers. Next, Section 6.2 introduces so-called selection strategies, which provide a heuristic approach to guiding the explainers during the search. Following that, Section 6.3 presents GreeDyCF, which is a low-complexity approach that uses a greedy heuristic for identifying counterfactual explanations. The subsequent section (Section 6.4) introduces CoDy, which is a sophisticated search approach that builds upon past findings in reinforcement learning and factual explanation methods.

6.1 Search Space

As outlined in Section 5.3, a counterfactual example $\mathcal{X}_{\varepsilon_i}$ that explains the prediction y_{ε_i} for the occurence of a potential future link ε_i in the dynamic graph $\mathcal{G}$ could be any combination of past events $\mathcal{X}_{\varepsilon_i} \subseteq (\mathcal{G}(t_i) \setminus \varepsilon_i)$. Let S_{ε_i} denote the set containing all possible combinations of past events[1]:

$$S_{\varepsilon_i} = \bigcup_{k=0}^{|(\mathcal{G}(t_i)\setminus\varepsilon_i)|} \binom{(\mathcal{G}(t_i) \setminus \varepsilon_i)}{k} \tag{6.1}$$

Following the binomial theorem, for the $n = |(\mathcal{G}(t_i) \setminus \varepsilon_i)|$ past events, the complexity of enumerating over all combinations by brute force is $O(2^n)$, making a brute force approach highly computationally expensive, even at small n. Thus, the search space for possible counterfactual examples needs to be explored by more efficient means. As a first step towards making the search space more manageable, spatial and temporal constraints on the search space are introduced.

6.1.1 Constraints on the Search Space

A spatial and a temporal constraint are applied to limit the size of the search space. The spatial constraint restricts the past events considered in the search to those in a temporal k-hop-neighborhood $N_k(\varepsilon_i)$ of the explained event ε_i. If the link prediction function f is a TGNN, k is set to the number of layers in the TGNN since contemporary TGNNs predominantly aggregate information from events that occurred within the temporal k-hop-neighborhood [74]. As a temporal constraint, only the m_{max} most recent events that

[1] following the notation of [56]

fulfill the spatial requirements are considered in the search. Formally, let $C(\mathcal{G}, \varepsilon_i, k, m_{max})$ be the set of past events considered for the search, for which the spatial and temporal constraints mentioned above hold, then the constrained search space $\hat{S}_{\varepsilon_i}$ is given by:

$$\hat{S}_{\varepsilon_i} = \bigcup_{l=0}^{|C(\mathcal{G},\varepsilon_i,k,m_{max})|} \binom{C(\mathcal{G}, \varepsilon_i, k, m_{max})}{l} \tag{6.2}$$

While these constraints can reduce the search space substantially, the complexity remains exponential at $O(2^{m_{max}})$, but with a fixed upper bound in m_{max}. This means that even if only a moderate amount of $m_{max} = 32$ past events would be considered, more than four billion unique combinations of events remain. Thus, the search space is structured in a way that enables the discovery of counterfactual explanations in line with the objectives of the explainer (see Section 5.3).

6.1.2 Structuring the Search Space

Since traversing the full search space is infeasible, search algorithms are employed to search for counterfactual examples by successively building a partial search tree P. The partial search tree builds upon a complete tree structure described as search tree T. The search tree T and the partial search tree P are described as a set of nodes. A search tree node is denoted as $n_i \in T$ or $n_i \in P$ to avoid confusion with the unrelated nodes $v_i \in V$ in the graph. The search tree structures the search space in a way that facilitates finding simple counterfactual examples close to the root node of the search tree. Each node in the search tree is associated with a so-called perturbation set consisting of one of the combination of past events from the search space $\hat{S}_{\varepsilon_i}$. Formally, the search tree T describes a set of nodes, where each node $n_j \in T$ in this search tree is characterized by an associated perturbation set $s_j \in \hat{S}_{\varepsilon_i}$ from the search space. A node in the search tree represents the omission of the events in the perturbation s_j from the set of past events $\mathcal{G}(t_i)$. For a concise representation of the search tree, each node n_j is represented as a tuple of associated attributes:

$$n_j = (s_j, prediction_j, parent_j, children_j) \tag{6.3}$$

When a node is first initialized, some of these attributes may not yet have a value. To accommodate this, an absence of a value is denoted by the symbol *null*. The attributes denote different aspects of the node that are essential for the search:

- **Perturbation set** s_j: $s_j \in \hat{S}_{\varepsilon_i}$ denotes the set of perturbations associated with node n_j. The set of perturbations consists of a set of past events that are removed from the original input for the prediction $prediction_j$ associated with node n_j.

- **Prediction result** $prediction_j$: The prediction result $prediction_j \in (\mathbb{R} \cup \{null\})$ represents the prediction value that results from applying the link prediction function $prediction_j = f((\mathcal{G}(t_i) \setminus s_j), \varepsilon_i)$ for the explained event ε_i with the perturbed input $(\mathcal{G}(t_i) \setminus s_j)$.

- **Parent** $parent_j$: The parent attribute $parent_j \in (T \cup \{null\})$ establishes a connection between node n_j and the node from which it is derived. If n_j is the root node, the attribute is set to $null$.

- **Children** $children_j$: The set of child nodes $children_j \subset T$ contains all nodes directly connected to n_j and positioned one level below in the search tree.

As a starting point for the search, the root node n_{root} is initialized with the identity perturbation $s_{root} = \varnothing$. Since no perturbations are applied to the input at this stage, the root node is linked to the original prediction for the occurrence of the future link ε_i. The root node has a set of children, which themselves have their own children. The perturbation set associated with children nodes expands the perturbation set associated with their parent. Formally, for a for a parent node $n_{parent} \in T$ in the search tree, each child node $n_{child} \in children_{parent}$ is associated with a perturbation $s_{child} \in \hat{S}_{\varepsilon_i}$ that satisfies the following conditions:

$$s_{child} \in \hat{S}_{\varepsilon_i} \tag{6.4}$$

$$s_{parent} \subset s_{child} \tag{6.5}$$

$$|s_{child}| = |s_{parent}| + 1 \tag{6.6}$$

This means that the perturbation set s_{child} associated with the child node n_{child} has to be part of the search space (Equation 6.4). Additionally, the perturbation set s_{child} has to contain all the events that are part of the perturbation set of the parent node s_{parent} (Equation 6.5) and contain exactly one more event than s_{parent} (Equation 6.6).

This definition implies that the perturbations of the children represent an extension of their parent's perturbation set by one more event. Following a direct path from the root node to any node in the search tree can be interpreted as consecutively pruning one more event from the original input at each node along this path through the search tree. The depth of a node in the search tree, i.e., the length of the unique path from the root node to that node, corresponds to the number of events included in its associated perturbation.

Figure 6 illustrates a search tree. Following the leftmost path in the tree can be interpreted as initially removing event ε_1 from the past events and subsequently removing event ε_2. The figure visually represents the increasing size of perturbation sets associated with nodes at greater depths in the tree. Furthermore, it demonstrates how excluding the events in the perturbation sets from the input graph influences the prediction score. Generally, the set of all nodes in the search tree at depth k encompasses nodes linked to all possible candidate perturbations in the search space of size k.

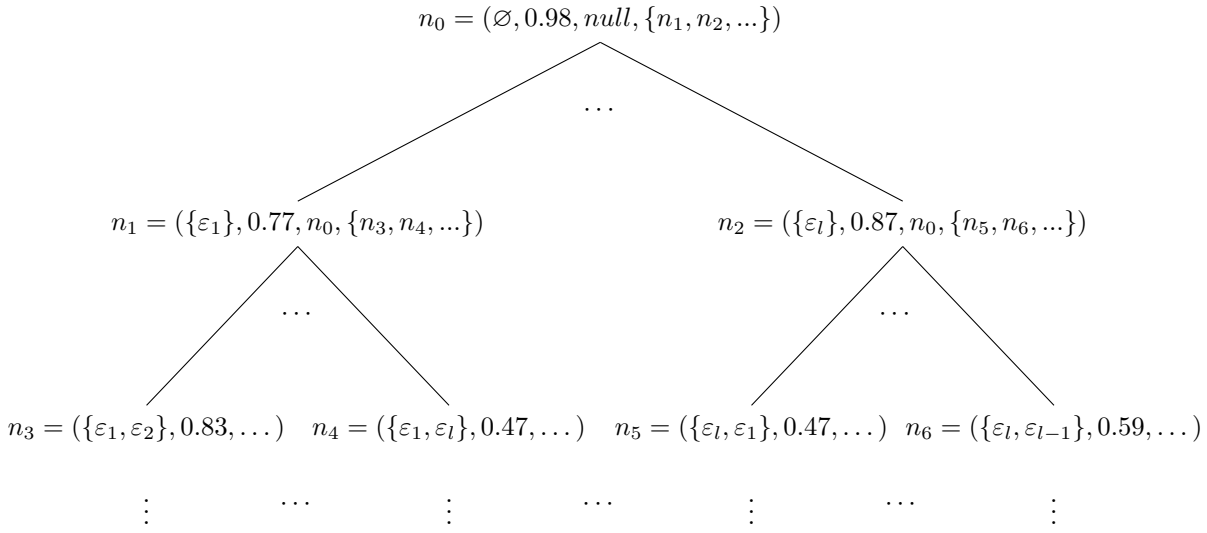


Figure 6: Example of a partial search tree for the explained event ε_i. Without loss of generality, it is assumed that the candidate events $C(\mathcal{G}, \varepsilon_i, k, m_{max})$ are denominated by $\varepsilon_1, \varepsilon_2, ..., \varepsilon_{l-1}, \varepsilon_l$.

Organizing the search tree in this manner reflects the objectives of the explainer (Section 5.3). The increasing size of the associated candidate perturbations with increasing depth is tethered to minimizing the complexity of the counterfactual explanation. It allows the exploration of low-complexity explanation candidates without expanding and traversing the partial search tree to large depths. The complexity of associated perturbation sets increases with the distance to the root node. The goal of maximizing discoveries is pursued in the consecutive pruning approach. This allows a search algorithm to follow paths on the search tree that are promising for leading to a counterfactual example.

GreeDyCF and CoDy build upon this search tree. Like the search space, the complete search tree T is too large to construct fully. Thus, the search approaches only construct a partial search tree P. In the search, the partial search tree P is iteratively expanded, starting from the root node. Expansion refers to the addition of a new node to the partial search tree. While both explainers share this framework for structuring the search space, the search strategies they employ vary significantly.

6.2 Selection Strategies

When a search algorithm traverses the partial search tree P, it may have to select between several previously unexplored child nodes as direction for growing the partial search tree. In this case, a so-called selection strategy is employed to guide the selection of the unexplored nodes. The selection strategies aim to improve the search performance by exploring promising alternatives first. Since there have not been any prior works on this, four different selection strategies are proposed and compared. Let $A = n_1, ..., n_k$ be the set of alternative nodes in the search tree that could be selected and that have not yet been associated with a prediction score, meaning they are unexplored. The sets of perturbations associated with each of these nodes differ by exactly one event. Let $\varepsilon_1, ..., \varepsilon_k$ be the single events that differ between the respective nodes. The selection strategies provide a ranking of the alternative nodes A based on these differing events. The higher a node is ranked, the more it is prioritized in the selection process.

- **Random**: The *random* selection strategy serves as the baseline strategy. It selects a node $n_j \in A$ at random.
- **Temporal**: The *temporal* selection strategy ranks the events by their recency. The most recent event is rated the highest, and the oldest event is rated the lowest.
- **Spatio-Temporal**: The *spatio-temporal* selection strategy ranks the events based on their spatial distance to the explained event ε_i. Lower distances are rated higher than larger distances. It ranks each event ε_j by the length of the shortest walk between any of the nodes involved in ε_j and any of those involved in ε_i. Events with the same spatial distance are ranked by their recency, just like in the *temporal* strategy. The *spatio-temporal* selection strategy thus represents an extension of the *temporal* selection strategy.
- **local-gradient**: The *local-gradient* selection strategy also operates on the events that differ between the perturbation sets. However, it does not use the spatio-temporal structure of the graph as basis for the ranking. Instead, it compiles a ranking of all alternative events A, based on the local gradient of the differing events. The local gradient of an event refers to the change in prediction that is achieved when omitting that single event from the original input graph. To get a ranking, the local gradient is calculated once for all candidate events $C(\mathcal{G}, \varepsilon_i, k, m_{max})$ before the first search iteration. To get the local gradient for the ranking, the link prediction function f is used to calculate a separate prediction $pred_j$ for each of the candidate events $\varepsilon_j \in C(\mathcal{G}, \varepsilon_i, k, m_{max})$ as:

$$pred_j = f((\mathcal{G}(t_i) \setminus \{\varepsilon_j\}), \varepsilon_i) \tag{6.7}$$

The prediction $pred_j$ is the prediction for the explained instance when event ε_j is omitted from the input. These predictions are then fed into the so-called Δ-function (see Equation 6.8), ranking the candidate events in decreasing order by how much the prediction is shifted.

The different selection strategies represent separate heuristic perspectives on the task of ranking alternative directions in the search tree. The *temporal* and *spatio-temporal* selection strategies leverage structural information. They are based on the intuition that events that are spatially and temporally close to the explained event are more important to the original prediction and, thus, more likely to be part of a counterfactual explanation. In comparison, the *local-gradient* strategy leverages the isolated impact of removing events separately for the ranking.

The Δ-function $\Delta(prediction_{orig}, prediction_j)$ that is used in the *local-gradient* selection strategy determines how much a prediction $prediction_j$ is shifted towards being classified differently from the original prediction $prediction_{orig}$. Recalling the definition of the classification of a prediction from Section 5.1, predictions that are classified differently from another have opposing signs, i.e., a positive and a negative prediction are classified differently. The higher the result of the Δ-function, the more the prediction $prediction_j$ suggests a different outcome in the link prediction task.

$$\Delta(prediction_{orig}, prediction_j) = \begin{cases} prediction_{orig} - prediction_j, & \text{if } prediction_{orig} \geq 0 \\ prediction_j - prediction_{orig}, & \text{else} \end{cases} \tag{6.8}$$

6.3 Greedy Explainer for Models on Dynamic Graphs using Counterfactuals

Greedy Explainer for Models on **Dy**namic Graphs using **C**ounter**f**actuals (GreeDyCF) provides an uncomplicated approach for searching for counterfactual examples in the search space. It aims to keep computations to a minimum in order to quickly conclude. The search approach revolves around a greedy strategy for traversing the search space.

The algorithm greedily follows the most promising path through the search tree, starting with the root node n_{root}. It iteratively expands the search tree along the nodes in the search tree, associated with the largest shift in predictions toward changing signs. This is in line with the iterative pruning logic underlying the structure of the search tree. The algorithm for GreeDyCF is depicted in Algorithm 1. It takes the link prediction model f, the input graph $\mathcal{G}$, the explained event ε_i, a selection strategy δ, and an adjustable

sampling parameter $l \in \mathbb{N}$ as inputs. The sampling parameter l determines how many nodes of the search tree GreeDyCF samples in each iteration. Before the first iteration, the original prediction is determined (line 1), the root node is initialized with the prediction attribute $prediction_{root}$ set to the original prediction $prediction_{orig}$ and without children (line 2), and the best node yet encountered n_{best} is set to the root node (line 3). Each iteration has three main steps: Sampling multiple child nodes (line 5), selecting the best child node from the sample (line 6), and checking how the selection affects the prediction (lines 7-14).

Algorithm 1: GreeDyCF search algorithm for counterfactual examples.

Input: Link prediction function f, input graph $\mathcal{G}$, explained event ε_i, selection strategy δ, number of nodes to sample in each iteration l

Output: best explanation found $\mathcal{X}$

1 $prediction_{orig} \leftarrow f(\mathcal{G}(t_i), \varepsilon_i)$;
2 $n_{root} \leftarrow (\varnothing, prediction_{orig}, null, \varnothing)$;
3 $n_{best} \leftarrow n_{root}$;
4 **while** s_{best} does not include all candidate events $C(\mathcal{G}, \varepsilon_i, k, m_{max})$ **do**
5 $\quad children_{best} \leftarrow$ set of l child node with highest rank according to δ, each child n_{child} initialized with prediction $prediction_{child} = f((\mathcal{G}(t_i) \setminus s_{child}), \varepsilon_i)$;
6 $\quad n_{best\_child} \leftarrow \arg\max_{n_j \in children_{best}} \Delta(prediction_{orig}, prediction_j)$;
7 $\quad$ **if** $\Delta(prediction_{best}, prediction_{best\_child}) > 0$ **then**
$\quad\quad$ /* n_{best_child} shifts the prediction further towards the opposite sign of the original prediction */
8 $\quad\quad n_{best} \leftarrow n_{best\_child}$;
9 $\quad\quad$ **if** $\Delta(prediction_{orig}, prediction_{best}) > |prediction_{orig}|$ **then**
$\quad\quad\quad$ /* s_{best} is a counterfactual example */
10 $\quad\quad\quad$ **break**;
11 $\quad\quad$ **end**
12 $\quad$ **else**
13 $\quad\quad$ **break**;
14 $\quad$ **end**
15 **end**
16 **return** s_{best}

A total of l child nodes are sampled in each iteration. The sampling process is guided by the selection strategy δ. From all possible child nodes, only the l that rank the highest according to δ are sampled. When sampling the child nodes, they are directly associated with their corresponding prediction attribute.

Next, out of the sampled children $children_{best}$, the child that shifts the prediction most towards changing signs is selected and assigned as best child node n_{best_child}. To infer how much each of the children shifts the prediction, the Δ-function that has been introduced in Section 6.2 is used. Formally, the best child node is determined as:

$$n_{best\_child} \leftarrow \underset{n_j \in children_{best}}{\arg\max} \; \Delta(prediction_{orig}, prediction_j) \tag{6.9}$$

To check how the selection affects the prediction, it is first checked whether the selected child node n_{best_child} shifts the prediction further than the best node found before n_{best} (Algorithm 1 line 7). If it does not, the search concludes because no direct improvement is possible. If it does shift the prediction further, then n_{best_child} becomes the new best node n_{best}. If the best child node constitutes a counterfactual example (Algorithm 1 line 9), the search concludes since it found a necessary counterfactual explanation.

In the end, the algorithm returns the perturbation set associated with the best node in the search tree n_{best} that has been encountered. This is either a counterfactual example or the perturbation set that results in the greatest non-counterfactual shift out of all perturbation sets that are associated with a node in the partial search tree.

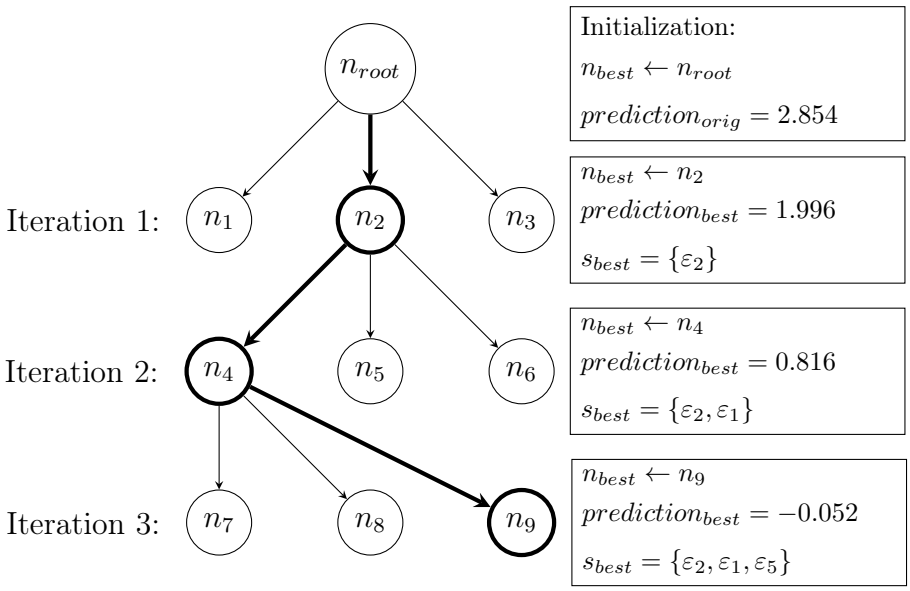


Figure 7: Example for the operation of the GreeDyCF approach.

Figure 7 shows a simplified schema for how GreeDyCF operates. In the figure, each iteration consists of sampling $l = 3$ past events using some selection strategy δ. After initializing the root node and the original prediction, node n_2 is selected in the first iteration as the new best node n_{best} from the sample. It is associated with the perturbation set $s_2 = \{\varepsilon_2\}$, resulting in a prediction of $prediction_2 = 1.996$. Throughout the search

procedure, the prediction associated with the best node gets lower with each iteration until it reaches a negative value in the third iteration. When this negative prediction is reached, the search concludes, returning the counterfactual example $\mathcal{X} = s_9 = \{\varepsilon_2, \varepsilon_1, \varepsilon_5\}$.

6.4 Counterfactual Explainer for Models on Dynamic Graphs

Counterfactual Explainer for Models on **Dy**namic Graphs (CoDy) uses a search algorithm that is mainly adapted from Monte Carlo Tree Search (MCTS) [31, 54]. The MCTS algorithm has been successfully applied to playing games [54] and planning problems [13]. Recently, it has also been used for factual explanations on static GNNs [74, 75] and TGNNs [67].

To accommodate the requirements of CoDy, the definition of the nodes in the search tree is extended by further attributes. Definition 6.10 replaces the initial Definition 6.3 of the tuple-representation for nodes in the search tree.

$$n_j = (s_j, prediction_j, parent_j, children_j, selections_j, score_j, selectable_j) \tag{6.10}$$

This extension adds the attributes $selections_j$, $score_j$, and $selectable_j$. The meaning of these attributes is as follows:

- **Selections** $selections_j$: The number of times this particular node has been selected in the search is denoted by $selections_j \in (\mathbb{N} \cup \{null\})$.
- **Score** $score_j$: The attribute $score_j \in (\mathbb{R} \cup \{null\})$ is used as a means to describe how promising node n_j is to lead to a counterfactual example.
- **Selectable** $selectable_j$: $selectable_j$ is a binary flag that determines whether node n_j can be chosen for further exploration. It is a binary value defined as $selectable_j \in \{0, 1\}$, where 1 signifies that n_j is selectable, and 0 that it is not.

Analogously to MCTS, the search algorithm of CoDy iteratively explores and expands a partial search tree using the results of past explorations and expansions as a guide on what to explore next [13]. The search operates in iterations, each iteration growing the partial search tree further. It concludes after a predefined number of maximum iterations it_{max} or when the entire search tree has been explored. At the end of the search, the algorithm selects the node associated with the minimal counterfactual example or a fallback option if no counterfactual example has been found. This fallback is introduced in Section 6.4.5. One search iteration consists of the same steps as the search iterations in MCTS [13]: Selection, Simulation, Expansion, and Backpropagation. However, the exact implementation of these steps differs from their MCTS counterparts.

Algorithm 2 provides a high-level overview of the CoDy search algorithm. The algorithm takes the link prediction function f, the dynamic graph $\mathcal{G}$, a potential future link ε_i, and a selection strategy δ as input. Additionally, the algorithm is provided with a maximum number of iterations it_{max}. The algorithm first computes the original prediction $prediction_{orig}$ and initializes the root node n_{root} of the search tree. The main search logic happens in iterations, illustrated by the while loop (lines 4-10). This loop concludes after it_{max} iterations or when the search tree is fully explored. A fully explored search tree is identified when the root node n_{root} is not selectable anymore. This may be the case when it is not possible to find new counterfactual examples. Each iteration starts with a selection process (line 5). This selection process traverses the partial search tree starting at the root node. This traversal ends when a node that has not yet been expanded is encountered. This node is then returned and selected. Next, the prediction associated with the perturbation of the selected node is computed in the simulation step (line 6). Subsequently, the node is expanded (line 7), and finally, the results are propagated upwards through the search tree (line 8), updating the attributes of the nodes with the new information that results from the previous steps. The search concludes by selecting the input perturbation that provides the best counterfactual explanation of the prediction among the input perturbations it explored (line 11). The inner workings of these steps are the subject of the following sections.

Algorithm 2: Search algorithm used by CoDy.

Input: Link prediction function f, input graph $\mathcal{G}$, explained event ε_i, selection strategy δ, maximum number of iterations it_{max}

Output: best explanation found

1 $prediction_{orig} \leftarrow f(\mathcal{G}(t_i), \varepsilon_i)$;
2 $n_{root} \leftarrow (\varnothing, null, null, \varnothing, 0, null, 1)$;
3 $it \leftarrow 0$;
4 **while** $it < it_{max}$ *and* n_{root} *is selectable* **do**
5 $\quad n_{selected} \leftarrow$ **select**(n_{root}, δ);
6 $\quad$ **simulate**$(n_{selected}, f, \mathcal{G}, \varepsilon_i)$;
7 $\quad$ **expand**$(n_{selected}, prediction_{orig})$;
8 $\quad$ **backpropagate**$(parent_{selected})$;
9 $\quad it \leftarrow it + 1$;
10 **end**
11 $n_{best} \leftarrow$ **select_best**(n_{root});
12 **return** s_{best}

6.4.1 Selection

The first step in each search iteration is the selection of a new node to expand. The selection procedure initiates from the root node and subsequently employs a recursive child selection policy to traverse the partial search tree until encountering a node suitable for expansion. A node $n_j \in P$ in the partial search tree qualifies for expansion if it has not yet undergone the expansion process, implying that it is not yet associated with child nodes ($children_j = \varnothing$) and it is not yet affiliated with a prediction score ($prediction_j = null$).

The child selection algorithm aims to balance the exploration of unknown parts of the search tree with the exploitation of already-known promising paths through the search tree. Given a node in the search tree n_j that has already been expanded, the selection procedure selects one of its child nodes. The selection hinges on a score $selection\_score(n_k)$ that is calculated for each selectable child $n_k \in selectable\_children(n_j)$ of node n_j:

$$selectable\_children(n_j) = \{n_k : n_k \in children_j,\ selectable_k = 1\} \tag{6.11}$$

A child node n_k is part of the selectable children $selectable\_children(n_j)$ of node n_j if it is marked as selectable, with a selectable attribute of $selectable_k = 1$. The child node n_k out of the selectable children that has the highest score $selection\_score(n_k)$ is selected. Formally, this is given as:

$$n_{best} \leftarrow \underset{n_k \in selectable\_children(n_j)}{\arg\max} selection\_score(n_k) \tag{6.12}$$

The selection score combines the exploitation score attribute $score_k$ associated with the node n_k with an exploration score $score_{exploration}(n_k)$, which is calculated based on the node's level of past exploration. It is defined as:

$$selection\_score(n_k) = \alpha * score_k + \beta * score_{exploration}(n_k) \tag{6.13}$$

Here, α and β are tunable hyperparameters that adjust the balance between the two terms. The exploitation term $\alpha * score(n_j)$ encourages selecting nodes already associated with a high exploitation score. In contrast, the exploration term $\beta * score_{exploration}(n_j)$ promotes the exploration of nodes with little past selections.

The exploration score is adapted from the 'Upper Confidence Bound 1' introduced in [4], which is popular across various MCTS implementations [31, 13]. For a node n_k with parent $n_j = parent_k$, the score is defined as:

$$score_{exploration}(n_k) = \sqrt{\frac{ln(selections_j)}{selections_k}} \tag{6.14}$$

This score $score_{exploration}(n_k)$ quantifies the uncertainty about selecting node n_k [4, 13]. A higher exploration score means that less is known about the potential impact on the prediction that follows from selecting this node. The less a node n_k is explored in comparison to the other children of its parent n_j, the higher the exploitation score for n_k.

In contrast to the exploration score, the exploitation score $score_k$ is not calculated during the selection step. Rather, this attribute is set at the expansion step of the CoDy search algorithm (see Section 6.4.3) and updated in the backpropagation (see Section 6.4.4).

If more than one child node has the same selection score, a selection strategy δ is used to select a child. This selection strategy can be any of the selection strategies introduced in Section 6.2. For example, the selection strategy is used when none of the children of a node have been expanded yet and are thus associated with the same score. The complete selection process is described in Algorithm 3, which recursively explores the search tree following the outlined procedure. The recursion ends when a node is encountered that has not yet been expanded (line 2). If multiple child nodes share the same highest selection score (line 6), the selection strategy δ is used to make the selection between the highest-ranking children.

Algorithm 3: Recursive selection algorithm.

```
1  Function select(n_j, δ) :
2      if n_j is not yet expanded then
3          return n_j;
4      end
5      n_best ← arg max_{n_k ∈ selectable_children(n_j)} selection_score(n_k);
6      if there is more than one child with the highest selection score then
7          n_best ← highest ranking child node according to selection strategy δ;
8      end
9      return select(n_best);
10 end
```

6.4.2 Simulation

The next step after selecting a node n_j is referred to as simulation. This step consists of making a call to the link prediction function f to infer the prediction associated with the selected node. The prediction $prediction_j$ associated with the selected node n_j is calculated as the prediction for the explained event ε_i when the events in the perturbation set s_j associated with node n_j are removed from the set of past events $\mathcal{G}(t_i)$ in the original input:

$$prediction_j = f((\mathcal{G}(t_i) \setminus s_j), \varepsilon_i) \tag{6.15}$$

This procedure is outlined in Algorithm 4. It shows how the simulate function infers the score associated with the selected node.

Algorithm 4: Algorithm for simulating the link prediction on the selected node.

```
1 Function simulate(n_j, f, G, ε_i) :
2 |   prediction_j ← f((G(t_i) \ s_j), ε_i);
3 |   return prediction_j;
4 end
```

6.4.3 Expansion

Following the simulation step, the expansion step involves updating the attributes of the selected node n_j. The exploitation score $score_j$ is initialized as the relative magnitude of the shift between the original prediction score $prediction_{orig}$ and the prediction score linked to the expanded node $prediction_j$. If this value is negative, the exploitation score is set to 0. Formally, $score_j$ is initialized as:

$$score_j \leftarrow \max\left(0, \frac{\Delta(prediction_{orig}, prediction_j)}{|prediction_{orig}|}\right) \tag{6.16}$$

The Δ-function introduced in Section 6.2 is reused here. If the score defined in 6.16 is greater than 1, that signifies that the link prediction associated with node n_j is classified differently from the original link prediction, making s_j a counterfactual example.

In addition to calculating the exploitation score, the expansion step also adds a set of children to the selected node, which expands the partial search tree. As discussed in Section 6.1, all child nodes n_k of the expanded node n_j have to satisfy the conditions described in Equations 6.4, 6.5, and 6.6. That is, the perturbation sets associated with the child nodes have to extend the perturbation set associated with the expanded node

by exactly one event. Thus, the set of child nodes to the expanded node n_j is initialized as:

$$children_j \leftarrow \{(s_k, null, n_j, \varnothing, 0, null, 1) : s_k \in \hat{S}_{\varepsilon_i} \text{ with } |s_k| = |s_j| + 1, s_j \subset s_k\} \quad (6.17)$$

The child nodes are initialized with *null* values for their prediction and score attributes and an empty set of child nodes. These attributes are only assigned once any of these child nodes are selected and expanded themselves.

The expansion step is summarized in Algorithm 5. After assigning the initial score to the selected node (line 3), the function checks whether the prediction associated with the node is counterfactual to the original prediction (line 4). If so, the algorithm concludes and sets the selectable flag $selectable_j$ to 0, which prevents future selections of node n_j. It would not make sense to select n_j again, as it already constitutes a counterfactual example. Further deepening the search tree from node n_j could only encounter counterfactual examples with higher complexity, violating the objective of minimizing complexity. In the case that the prediction associated with the selected node n_j is not counterfactual to the original prediction, the child nodes $children_j$ are initialized (line 8). If there are no child nodes, the node n_j is marked as not selectable (line 10).

Algorithm 5: Function for expanding the selected node.

```
1  Function expand(n_j, prediction_orig) :
2      selections_j ← 1;
3      score_j ← max(0, Δ(prediction_orig, prediction_j) / |prediction_orig|);
4      if score_j > 1 then
5          selectable_j ← 0;
6          Add n_j to a list of counterfactual examples cf_examples;
7      else
8          children_j ← {(s_k, null, n_j, ∅, 0, null, 1) : s_k ∈ Ŝ_ε_j with |s_k| = |s_j| + 1, s_j ⊂ s_k};
9          if children_j = ∅ then
10             selectable_j ← 0;
11         end
12     end
13 end
```

6.4.4 Backpropagation

Backpropagation is the last step in each search iteration. It serves to update information in the search tree starting at the parent n_p of the expanded node n_j, recursively traversing

backward through the search tree until the root node is reached. The backpropagation increments the number of selections by one, updates the exploitation score, and checks if the node is still selectable.

The exploitation score $score_j$ of a node n_j is updated to estimate the potential for finding a counterfactual example among its descendants. It is thus calculated as the weighted average of the exploitation scores of its children and the initial exploitation score associated with n_j itself. The initial exploitation score results from the expansion step and is calculated according to Equation 6.16. Thus, the update is computed as follows:

$$score_j \leftarrow \frac{\max\left(0, \frac{\Delta(prediction_{orig}, prediction_j)}{|prediction_{orig}|}\right) + \sum_{n_k \in children_j}(score_k * selections_k)}{selections_j} \tag{6.18}$$

To estimate the potential for finding a counterfactual example among the descendants of node n_j, the exploitation scores of each child node n_k is weighted with its number of selections $selections_k$. This follows the assumption that the more a node is selected, the more accurate its estimation should become.

Algorithm 6 presents the full backpropagation procedure. For easier readability, the update of the exploitation score is presented over lines 6-8. The backpropagation also checks if any of the child nodes is selectable. If no child node is selectable, the partial search tree cannot be expanded from any of the descendant nodes. Thus, the node is marked as not selectable itself (line 10). The backpropagation concludes when it reaches the root node (line 2).

Algorithm 6: Backpropagation function that recursively updates the information of nodes in the search tree.

```
1  Function backpropagate(n_j) :
2      if n_j = null then
3          return;
4      end
5      selections_j ← selections_j + 1;
6      score_j ← max(0, Δ(prediction_orig, prediction_j) / |prediction_orig|);
7      score_j ← score_j + Σ_{n_k ∈ children_j}(score_k * selections_k);
8      score_j ← score_j / selections_j;
9      if No child in children_j is selectable then
10         selectable_j ← 0;
11     end
12 end
```

6.4.5 Explanation Selection and Fallback

After all search iterations have concluded, the node in the search tree associated with the best counterfactual explanation is selected. This is trivial when exactly one counterfactual example has been encountered during the search. If multiple counterfactual examples were encountered, the best of them gets selected according to the search objectives. When the search did not find any counterfactual examples, a fallback strategy is invoked that provides the next best explanation.

Selecting the best node associated with a counterfactual example amongst several candidate nodes $cf\_examples$ involves rating the candidates according to the objective of minimizing complexity. Accordingly, only the examples $n_j \in cf\_examples$ with the least number of associated perturbations s_j are considered:

$$smallest\_examples \leftarrow \underset{n_j \in cf\_examples}{\arg\min} |s_j| \tag{6.19}$$

Since there may be multiple nodes in the set of nodes associated with the smallest examples $smallest\_examples$, a further step has to be made. Thus, the node from the $smallest\_examples$ that has the highest exploitation score is selected. Ordering the nodes in this manner ensures that the selected counterfactual example achieves its counterfactual status with a larger margin than the other smallest examples. Formally, the best node n_{best} is set as:

$$n_{best} \leftarrow \underset{n_j \in smallest\_examples}{\arg\max} score_j \tag{6.20}$$

In the case that no counterfactual example could be found during the search, CoDy provides a fallback explanation. This fallback is the perturbation set associated with the node n_j from the partial search tree P, which has the prediction $prediction_j$ that is shifted the most towards the opposite sign of the original prediction. Formally, the node in the search tree that is associated with the largest shift n_{best} is defined as:

$$n_{best} \leftarrow \underset{n_j \in P}{\arg\max} \Delta(prediction_{orig}, prediction_j) \tag{6.21}$$

These strategies are part of Algorithm 7, which shows the selection process. If possible, the algorithm selects the node associated with the smallest counterfactual example (lines 3-5). If not, it employs the fallback strategy to select a node accordingly (lines 6-7).

Algorithm 7: Algorithm for selecting the best counterfactual example, or the example that comes closest to being counterfactual.

Function ***select_best***(n_{root}) **:**
  $P \leftarrow$ set of all nodes in the partial search tree starting from root node n_{root};
  if *the search did find counterfactual examples* $cf\_examples$ **then**
    $smallest\_examples \leftarrow \arg\min_{n_j \in cf\_examples} |s_j|$;
    $n_{best} \leftarrow \arg\max_{n_j \in smallest\_examples} score_j$;
  else
    $n_{best} \leftarrow \arg\max_{n_j \in P} \Delta(prediction_{orig}, prediction_j)$;
  end
  return n_{best};
end

6.4.6 Search Optimizations

While the CoDy search strategy, as it is presented in the previous sections, already provides a good basis for searching for counterfactual explanations, the procedure is further optimized. CoDy employs dynamic constraints on the search tree to minimize the complexity of explanations. Additionally, caching is used to minimize redundant computations, and an approximation approach is utilized to substantially speed up the inference time of the link prediction function.

The first optimization leverages the objective of minimizing the complexity of the counterfactual example. When the search algorithm encounters a node n_j for which the associated perturbation set s_j constitutes a counterfactual example, any other node n_k associated with a perturbation set s_k containing more events than s_j would be considered worse than n_j. Thus, the search should not explore nodes in the search tree associated with a perturbation set that has a higher complexity than s_j. The complexity of a perturbation set s_j is measured by the number of events it contains $|s_j|$. CoDy implements this dynamic constraint on exploring the search tree by modifying the search tree once a node n_j associated with a counterfactual example is expanded. This is done by marking all nodes $n_k \in T$ with $|s_k| = |s_j| + 1$ as not selectable ($selectable_k = 0$) and then updating the selectability attribute of all nodes n_l with lower or equal complexity $|s_l| \leq |s_j|$. To update the selectability attribute of a node n_l, the selectability attribute of all its child nodes $children_l$ is checked, and if none of the child nodes remain selectable, the selectability attribute $selectable_j$ of node n_j is set to 0 itself. This is the same procedure that is also used in the backpropagation step. Additionally, from thereon, when a node n_k with the same complexity as n_j is selected and expanded, it is automatically marked as non-selectable to prohibit the search algorithm from exploring its children.

The other optimizations do not alter the search algorithm but provide means to speed up the runtime of the search significantly. Making calls to the link prediction function can be costly since models on dynamic graphs, like TGNNs, tend to have a high computational cost. That is because they need to process all past events to make a prediction. Thus, the optimizations aim to prevent redundant calls to such models and speed up inference when calling the link prediction function.

By design, the search tree can have multiple nodes that are associated with the same set of perturbations. For instance, the perturbation set $s_j = \{\varepsilon_3, \varepsilon_5\}$ can be encountered when first selecting ε_3 and then selection ε_5, or the other way around. While this results in two distinct nodes in the search tree, the prediction associated with both of them is the same. Thus, CoDy caches the prediction results associated with a perturbation the first time a node with that perturbation is expanded. When another node that shares the same perturbation set is expanded in a later iteration, it is associated with the cached prediction instead of calling the link prediction function again.

The last optimization concerns the calls to the link prediction function. For many of the state-of-the-art TGNN models operating on CTDGs [51, 55], the runtime for predicting a potential future link ε_i is dependent on the how many past events $\mathcal{G}(t_i)$ there are in the input. Thus, reducing the number of past events also reduces the runtime of such a link prediction function. To harness this potential, CoDy employs a two-stage approximation-confirmation approach that improves runtimes while maintaining accurate predictions. When the link prediction function is called to get a prediction for the occurrence of a potential future link ε_i with input perturbations s_j, not all events are used for the initial inference of the prediction. The reduced set of past events for inferring the prediction with an input perturbation s_j consists of two parts: First, all events that happen prior to the earliest candidate event ε_{min} that is considered for the explanation. Second, all candidate events $C(\mathcal{G}, \varepsilon_i, k, m_{max})$ that are not part of the input perturbation s_j.

The events prior to the earliest candidate event are included for an accurate basis of the dynamic graph. From the oldest candidate event onwards, only the candidate events are included because the candidate events are chosen in a way that aims to select all the recent events that can significantly influence the prediction, as outlined in Section 6.1.1. Thus, it can be assumed that the other events that take place after ε_{min} are of insignificant importance to the prediction. Hence, the omission of these events has only a minor effect on the output of the link prediction function.

Formally, the oldest candidate event $\varepsilon_{min} \in C(\mathcal{G}, \varepsilon_i, k, m_{max})$ is defined as:

$$\varepsilon_{min} = \underset{\varepsilon_j \in C(\mathcal{G}, \varepsilon_i, k, m_{max})}{\arg\min} t_j \tag{6.22}$$

Thus, the approximated prediction $prediction_l^{approximation}$ for the occurrence of the future link ε_i with input perturbation s_l is calculated as:

$$prediction_l^{approximation} = f(((\mathcal{G}(t_{min-1}) \cup C(\mathcal{G}, \varepsilon_i, k, m_{max})) \setminus s_l), \varepsilon_i) \tag{6.23}$$

The confirmation stage for the prediction is only invoked if the approximated prediction suggests that the input perturbation s_l constitutes a counterfactual example. In such cases, the prediction is confirmed by computing the exact score, using all past events besides the input perturbation for the prediction:

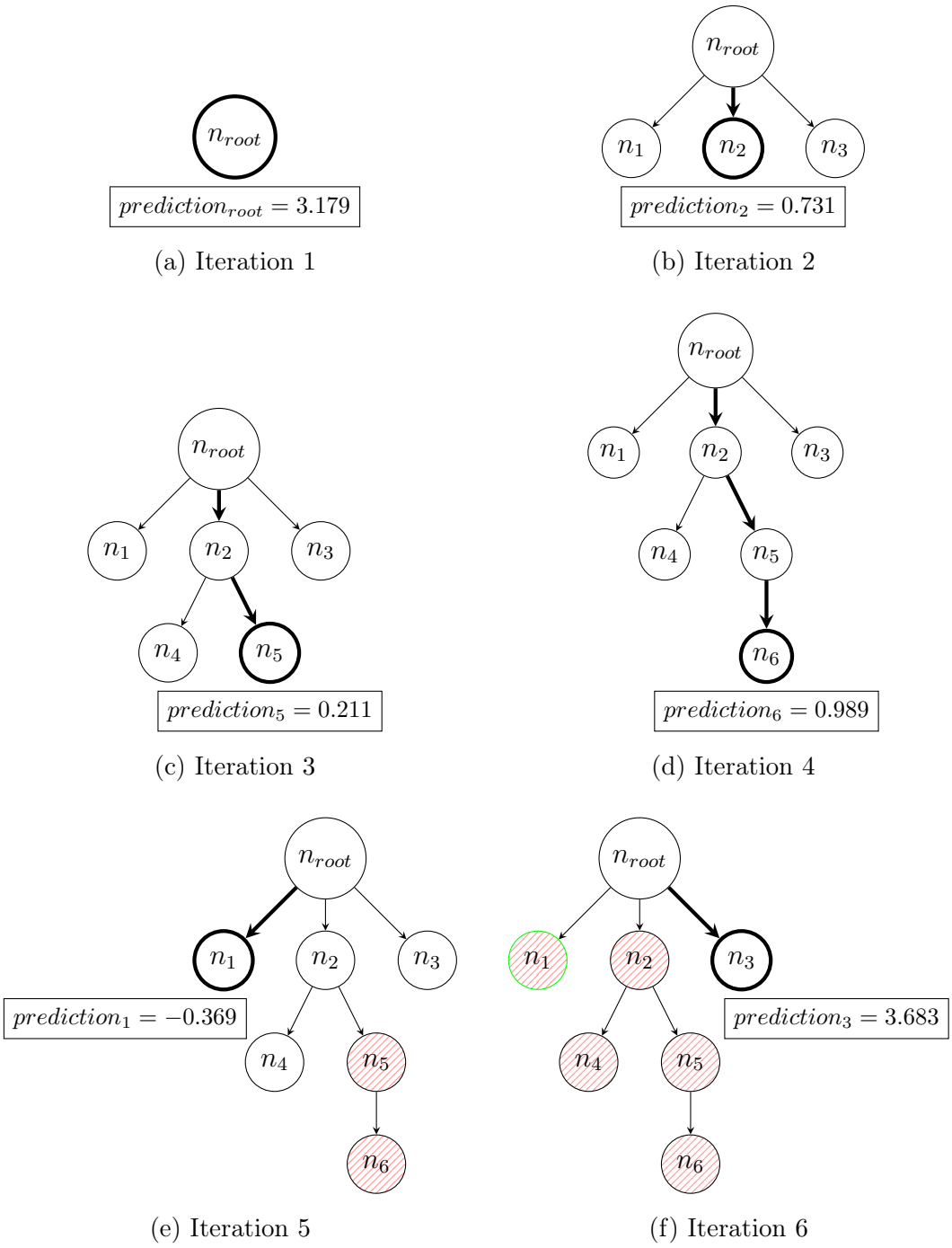
$$prediction_l^{confirmation} = f((\mathcal{G}(t_i) \setminus s_l), \varepsilon_i) \tag{6.24}$$

This two-stage approach substantially speeds up the search process through the approximation. Using the confirmation step keeps the results on potential counterfactual examples accurate.

6.4.7 Example of the Search Algorithm

Figure 8 exemplifies the operations of the CoDy algorithm. It depicts how the CoDy search algorithm operates and constructs the partial search tree. In the depicted case, the candidate events $C(\mathcal{G}, \varepsilon_i, k, m_{max})$ only consist of three past events $\varepsilon_1, \varepsilon_2, \varepsilon_3$. Following along the search algorithm, in the first iteration the root node is selected. Simulating on the root node produces the original prediction score, which is 3.179. This score means that the potential future link ε_i is classified to occur. The expansion step in the first iteration adds new child nodes to the root, one for each of the candidate events. Let $s_1 = \{\varepsilon_1\}, s_2 = \{\varepsilon_2\}, s_3 = \{\varepsilon_3\}$. Since none of the child nodes of the root have been explored to this point, the second iteration employs the selection strategy δ to select node n_2. After simulation, this node is expanded, adding two child nodes associated with the input perturbations $s_4 = \{\varepsilon_1, \varepsilon_2\}$, and $s_5 = \{\varepsilon_2, \varepsilon_3\}$. Iteration 3 selects node n_5. The expansion only adds a single child node associated with $s_6 = \{\varepsilon_1, \varepsilon_2, \varepsilon_3\}$. In iteration 4, the newly added child node n_6 is selected. Since it has no children, backpropagation marks the node as no longer selectable. The same applies to its parent node n_5. In iteration 5, the exploration score associated with the unexplored children of the root node is so large that n_2 is no longer selected over the unexplored alternatives. Thus, δ is applied, resulting in the selection of node n_1. The simulation of n_1 yields a negative score, meaning that its associated perturbation set constitutes a counterfactual example. Following the optimization of dynamic constraints on the search space, all nodes associated with a perturbation set that contains more than one element are marked as no longer selectable. In this instance, this applies to node n_4. Marking node n_4 as not selectable also makes

node n_2 no longer selectable since none of its children remain selectable. Thus, in the final iteration, node n_3 is selected as the only option. After this iteration, none of the nodes remain selectable, meaning that the search tree is fully explored and the search can conclude. As node n_1 is the only node associated with a counterfactual example, it is selected as the best node, and the perturbation set $s_1 = \{\varepsilon_1\}$ is returned as the minimal counterfactual example.



(a) Iteration 1

(b) Iteration 2

(c) Iteration 3

(d) Iteration 4

(e) Iteration 5

(f) Iteration 6

Figure 8: Schematic depiction of the iterative expansion of the partial search tree.

7 Evaluation

This chapter provides a comprehensive evaluation of GreeDyCF and CoDy. To assess the performance and effectiveness of both methods in various scenarios, a series of experiments is conducted in different settings and on multiple datasets, comparing the proposed approach against a baseline explanation method. This chapter is partitioned into two sections. First, Section 7.1 details the experimental setup for the evaluation. Second, Section 7.2 presents and discusses the results of the experiments.

7.1 Experimental Setup

To ensure the replicability of the results of the evaluation, this section details the experimental setup used for evaluating GreeDyCF and CoDy. First, the datasets are introduced (Section 7.1.1). Then, the explained target model and its configuration are detailed in Section 7.1.2. Section 7.1.3 details the explainers compared in the evaluation. Subsequently, Section 7.1.4 outlines the metrics that are used to evaluate and compare the explanation approaches. After a brief discussion of the physical evaluation infrastructure (Section 7.1.5), the different experimental settings are outlined in Section 7.1.6.

7.1.1 Datasets

The explainers are evaluated on three different datasets. These datasets are diverse in terms of their size, their structure, and the temporal density of events, aiming to verify that the explanation approaches perform similarly on different datasets.

The first two datasets come from an online social network among students of the University of California at Irvine (UCI) [33]. The first of these is called UCI-Messages. In this dynamic graph, nodes represent students and edges private messages between these students. No node or edge features are included in the dataset.

UCI-Forums is the second dataset. The nodes in this bipartite dynamic graph represent students and forums. Each edge represents a post that a student made to a specific forum. Neither the nodes nor the edges are associated with features.

The third dataset is called Wikipedia [32] and consists of events representing edits of Wikipedia pages. The edits are represented as attributed links between users and pages. Thus, the dataset is a bipartite graph. It encompasses the 1,000 most edited pages and a total of 8,227 users. The edges are associated with attributes that represent the associated edit texts as Linguistic Inquiry and Word Count (LIWC) [46] feature vector. This vector encodes a quantitative analysis of the text in the page edit [46].

Table 3 provides an overview of these datasets. The two datasets from the UCI social network cover a longer timespan and are more sparse in the temporal dimension than the Wikipedia dataset. Another substantial difference is that, in contrast to the other datasets, the UCI-Messages dataset contains a relatively high number of unique edges compared to the number of total edges. Furthermore, the UCI-Messages and UCI-Forums datasets have proportionally fewer multi-edges compared to the Wikipedia dataset. An aspect that these datasets have in common is that they exclusively consist of addition events, meaning there only exist node and edge addition events but no deletion or feature update events.

Dataset	# Nodes	# Edges	# Unique Edges	Timespan	Graph Type
UCI-Messages [33]	1,899	59,835	20,296	196 days	Unipartite
UCI-Forums [33]	1,421	33,720	7,089	165 days	Bipartite
Wikipedia [32]	9,227	157,474	18,257	30 days	Bipartite

Table 3: Statistics for datasets. Table adapted from [48].

7.1.2 Target Model

The target model that is explained in all experiments is the dynamic graph model TGN [51]. TGN is a TGNN for CTDGs that was introduced in Section 2.4. TGN is used as the target model since it is popular within research [55] and achieves state-of-the-art performance in different tasks on dynamic graphs [51, 55]. To its users, the model mostly remains a black-box with little transparency over what leads to a specific prediction.

For the experiments, the TGN model is configured with a GRU as memory updater, graph attention as an embedding module, last message aggregation, and the identity memory function. This corresponds to the TGN-attn configuration presented as the default configuration by the authors of TGN [51]. The batch size is set to 32 events. For training, the data is split into four temporal sections. The first 20% of the data are processed but not used for training. The following 50% of events are used in the training. The remaining 30% are split into equal parts for validation and testing purposes. After training for 30 epochs with early stopping on the model accuracy, the TGN model archives a high average precision, as shown in Table 4.

Dataset	UCI-Messages		UCI-Forums		Wikipedia	
Setting	Transductive	Inductive	Transductive	Inductive	Transductive	Inductive
TGN-attn	85.86	83.26	92.72	89.39	97.80	97.39

Table 4: Average Precision in percent for future link prediction on the UCI-Messages, UCI-Forums, and Wikipedia dataset. The inductive setting consists of predicting future links between nodes that are seen during training. The transductive setting consists of predicting future links between nodes that are not observed during training.

7.1.3 Explainers

As there are not yet any counterfactual explanation methods for dynamic graph models, CoDy and GreeDyCF are evaluated against T-GNNExplainer [67] since this explainer also aims to explain predictions on CTDGs. TempME [15], the only other explanation method that targets CTDGs, is not considered because, as of the moment of writing, the authors have yet to release the code for this explainer.

Comparing factual and counterfactual explanation methods is a challenge, as these methods produce substantially different explanations [58]. Consequently, the assessment of explanatory performance is conducted separately for necessity (see Section 7.2.1) and sufficiency aspects (see Section 7.2.2), allowing for an in-depth investigation into both the factual and counterfactual dimensions of the explanations.

If not explicitly mentioned, all experiments in this evaluation are conducted with the same settings for the different explainers. The settings for the explainers are chosen to allow for a fair comparison between explanation methods while maintaining a reasonable runtime. Furthermore, the parameters for CoDy are not yet optimized. Tuning these parameters is left to future work.

The T-GNNExplainer is trained and configured as described by its authors [67]. The rollout number is fixed at 500 iterations, and the number of candidate events is limited to 30. The original implementation is used, including all the modifications the authors made to the target model TGN that are necessary for T-GNNExplainer to work properly.

For GreeDyCF, the upper limit on the candidate events m_{max} is set to 64. The sample size l is configured to 10, meaning that up to 10 events are sampled to expand the search tree in each iteration.

For CoDy, the upper limit on the candidate events m_{max} is also set to 64. The maximum number of iterations it_{max} is 300. To balance the exploration and exploitation of the search tree, the hyperparameters α and β are set to $\alpha = 2$ and $\beta = 1$.

7.1.4 Metrics

The choice of appropriate metrics is crucial for evaluating the effectiveness of explanation methods to allow for a comprehensive and fair assessment of their performance. As discussed in Sectio 5.3, the primary objectives of the explainers are maximizing discoveries and minimizing complexity. To judge how well CoDy and GreeDyCF achieve these objectives, a diverse array of different metrics is applied.

The fidelity metric is employed as a measure that evaluates how well explanation methods identify important input features [73]. Prior research has established a variety of definitions for the fidelity metric [73, 39, 67, 3, 49]. Different formulations exist that target either the explained phenomenon or the explained model [3]. When targeting the phenomenon, fidelity is assessed in relation to the ground-truth concerning the occurrence of the future link; conversely, when targeting the model, fidelity is measured in reference to the prediction of the link prediction function [3]. Since the aim of this thesis is to explain models, not the data, the fidelity metric is calculated regarding the model. Specifically, this work adapts the scores fid_- and fid_+ [3, 73] to models on CTDGs:

$$fid_- = 1 - \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(p(f(\mathcal{G}(t_i)), \varepsilon_i) = p(f(\mathcal{X}_{\varepsilon_i}, \varepsilon_i))) \tag{7.1}$$

$$fid_+ = 1 - \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(p(f(\mathcal{G}(t_i)), \varepsilon_i) = p(f((\mathcal{G}(t_i) \setminus \mathcal{X}_{\varepsilon_i}), \varepsilon_i))) \tag{7.2}$$

Here, $\varepsilon_1, ..., \varepsilon_N$ are possible future links and $\mathcal{X}_{\varepsilon_1}, ..., \mathcal{X}_{\varepsilon_N}$ are the respective explanations for the predictions of their occurrences. The indicator function $\mathbb{1}(a = b)$ returns 1, if a is equal to b, else it return 0.

The fid_- score measures how well the explanation captures the relevant past events that lead the dynamic graph model to its prediction. It achieves this by comparing the original prediction of the targeted model on the full input graph $\mathcal{G}(t_i)$ with the prediction using only the events in the explanation $\mathcal{X}_{\varepsilon_i}$ as input graph. Thus, it measures the sufficiency of the explanations and is most useful for judging the quality of factual explanations [3]. The fid_- score can be interpreted as the fraction of explanations that are sufficient on their own to produce the original prediction. More relevant to counterfactual explanations is the fid_+ score, which measures whether the prediction changes when the explanation is removed from the set of past events. It compares the prediction on the full input graph $\mathcal{G}(t_i)$ with the prediction using the full input graph without the events in the explanation $(\mathcal{G}(t_i) \setminus \mathcal{X}_{\varepsilon_i})$ as input. Hence, it provides insights into the necessity of the explanations [3, 58]. Recalling the definition of a counterfactual example from Section 5.2, the fid_+ score represents the proportion of cases in which the explainer manages to find a necessary

explanation in the form of a counterfactual example. Therefore, the fid_+ metric provides a measure for the objective of maximizing discoveries. Both types of fidelity scores take on values between 0 and 1, with higher values being preferable over lower ones.

To bring the fid_+ and the fid_- score into a single score that characterizes both sufficiency and necessity requirements, the characterization score *char* proposed by Amara et al. [3] is adopted. It is defined as the weighted harmonic mean of the two fidelity scores:

$$char = \frac{w_+ + w_-}{\frac{w_+}{fid_+} + \frac{w_-}{fid_-}} \tag{7.3}$$

where w_+ and w_- are weights for fid_+ and fid_- that allow putting more emphasis on either sufficiency or necessity. To make a fair comparison between the counterfactual explainers proposed in this thesis and the factual baseline explainer, the weights are set to $w_+ = w_- = 0.5$. The characterization score *char* takes on values between 0 and 1, where larger values indicate better performance.

The sparsity metric *sparsity* is a widely used tool to gauge the complexity of explanations [73, 3, 49], which addresses the second objective of minimizing complexity.

$$sparsity = \frac{1}{N} \sum_{i=1}^{N} \frac{|\mathcal{X}_{\varepsilon_i}|}{|C(\mathcal{G}, \varepsilon_i, k, m_{max})|} \tag{7.4}$$

The explanations over which the metric is calculated are denoted by $\mathcal{X}_{\varepsilon_1}, ..., \mathcal{X}_{\varepsilon_N}$. The sparsity metric measures the ratio between the past events in the explanation $\mathcal{X}_{\varepsilon_i}$ and all past events considered as candidates for the explanation $C(\mathcal{G}, \varepsilon_i, k, m_{max})$. It takes on values between 0 and 1, with lower values being considered better than higher values.

Finally, the Jaccard similarity [26] $J(\mathcal{X}^j_{\varepsilon_i}, \mathcal{X}^k_{\varepsilon_i})$ is used to analyze the similarity between the explanations produced by different explanation methods j and k. The Jaccard similarity of two explanations $\mathcal{X}^j_{\varepsilon_i}$ and $\mathcal{X}^k_{\varepsilon_i}$ for the occurrence of the future link ε_i is defined as:

$$J(\mathcal{X}^j_{\varepsilon_i}, \mathcal{X}^k_{\varepsilon_i}) = \frac{|\mathcal{X}^j_{\varepsilon_i} \cap \mathcal{X}^k_{\varepsilon_i}|}{|\mathcal{X}^j_{\varepsilon_i} \cup \mathcal{X}^k_{\varepsilon_i}|} \tag{7.5}$$

The Jaccard similarity measures the similarity between two explanations by comparing the intersection of the explanations with the union of the explanations. The intersection of the explanation $\mathcal{X}^j_{\varepsilon_i} \cap \mathcal{X}^k_{\varepsilon_i}$ comprises the events included in both explanations, whereas the union $\mathcal{X}^j_{\varepsilon_i} \cup \mathcal{X}^k_{\varepsilon_i}$ includes the events that are present in either one or both explanations.

To get a single score that encompasses the entire evaluation sample, the overall Jaccard similarity $J_{j,k}$ for the explanations $\mathcal{X}^j_{\varepsilon_1}, ..., \mathcal{X}^j_{\varepsilon_N}$ and $\mathcal{X}^k_{\varepsilon_1}, ..., \mathcal{X}^k_{\varepsilon_N}$ produced by the explainers j and k is calculated as:

$$J_{j,k} = \frac{1}{N} \sum_{i=1}^{N} J(\mathcal{X}^j_{\varepsilon_i}, \mathcal{X}^k_{\varepsilon_i}) \tag{7.6}$$

7.1.5 Infrastructure

The experiments are conducted on a high-performance computing cluster to ensure replicable results. Each experiment is run on a machine with an Intel Xeon Gold 6230 CPU, 16GB of RAM storage, and an NVIDIA Tesla V100 SXM2 GPU with 32GB of VRAM.

7.1.6 Experiments

Experiments are conducted for explanations of predictions on the UCI-Messages, UCI-Forums, and Wikipedia datasets. Separate experiments are conducted for future links that are correctly predicted to occur by the targeted model and for those that have wrong predictions about their occurrences. This distinction is made because explanations of correct predictions can be substantially different from those of wrong predictions [3]. In the following, the combination of a dataset and the evaluation of explanations for either correct or wrong predictions is referred to as an experimental setting. Each experiment consists of explaining 200 predictions using one of the explanation methods. The same predictions are explained across different the evaluated explanation approaches. Both GreeDyCF and CoDy are evaluated with the selection strategies *random*, *temporal*, *spatio-temporal*, and *local-gradient* (see Section 6.2) separately. When discussing the results that one of the explanation methods achieved in combination with a selection strategy, the combination is referred to as "Explainer-*Selection-Strategy*", for example, CoDy-*temporal* for the CoDy explainer paired with the *temporal* selection strategy.

7.2 Results

This section presents and discusses the results of the evaluation. To take a look at the explanatory capabilities of the compared explainers, the produced explanations are first evaluated in terms of necessity (Section 7.2.1), then in terms of their sufficiency (Section 7.2.2), before taking a convergent perspective onto both necessity and sufficiency aspects of the explanations (Section 7.2.3). After the analysis of the explanatory performance, other aspects of the explanation approaches are compared. Section 7.2.4 takes a look at the runtime of the approaches. In Section 7.2.5, the influence of search iterations on the

performance of CoDy is evaluated in detail. Next, Section 7.2.6 presents and discusses the degree of similarity among the explanations produced by the evaluated explainers. Finally, Section 7.2.7 studies the effects of the selection strategy on the GreeDyCF and CoDy approaches.

7.2.1 Neccesity of Explanations

Explanations constitute counterfactual examples when they include the information that is necessary for the original prediction of the target model. As previously discussed, the fid_+ score provides a measure for the degree to which an explainer discovers counterfactual examples, which captures the necessity. Table 5 provides an overview of the fid_+ scores across the different experimental settings. In all but one setting, the CoDy approach with either the *spatio-temporal* or the *local-gradient* selection strategies performs best. There exists a pattern that CoDy-*spatio-temporal* performs better than CoDy-*local-gradient* when explaining correct predictions on any dataset, while CoDy-*local-gradient* outperforms CoDy-*spatio-temporal* whenever the focus is on wrong predictions. GreeDyCF-*spatio-temporal* performs best for correct predictions on the UCI-Messages and the UCI-Forums datasets. The *random* selection strategy is outperformed by the other selection strategies for both GreeDyCF and CoDy across all datasets, with the exception of CoDy-*temporal*, which performs slightly worse than CoDy-*random* for correct predictions on the Wikipedia dataset. This suggests that the selection strategies *temporal*, *spatio-temporal*, and *local-gradient* aid GreeDyCF and CoDy in finding necessary explanations. The advantage in terms of fid_+ scores achieved by these selection strategies over the *random* selection strategy is particularly pronounced for GreeDyCF.

Looking at the fid_+ metric in isolation ignores the objective of minimizing complexity. Thus, the sparsity of the explanations is considered as well. Table 6 shows the sparsity of the explanations provided by the different explainers. The table shows that both GreeDyCF and CoDy produce substantially more concise explanations than T-GNNExplainer. This underlines the complexity advantage of counterfactual explanations. The table also indicates that the different variants of GreeDyCF consistently achieve lower sparsity scores than the respective variants of CoDy. However, looking at these sparsity values without any reference to the fid_+ metric can be misleading since explanations that have minimal sparsity are only desirable if they also identify important information. This is not always the case. For instance, GreeDyCF-*random* achieves very low $sparsity$ scores in most settings. However, it also achieves a substantially lower fid_+ score than the other GreeDyCF variants.

	UCI-Messages		UCI-Forums		Wikipedia	
	Correct	Wrong	Correct	Wrong	Correct	Wrong
T-GNNExplainer	0.10	0.25	0.05	0.28	0.05	0.22
GreeDyCF-*random*	0.02	0.08	0.05	0.07	0.05	0.11
GreeDyCF-*temporal*	0.14	0.33	0.43	0.31	0.10	0.30
GreeDyCF-*spatio-temporal*	**0.20**	0.38	**0.46**	0.30	0.14	0.39
GreeDyCF-*local-gradient*	0.11	0.35	0.29	0.27	0.08	0.28
CoDy-*random*	0.11	0.36	0.32	0.31	0.14	0.43
CoDy-*temporal*	0.14	0.39	0.39	<u>0.38</u>	0.13	0.48
CoDy-*spatio-temporal*	**0.20**	<u>0.41</u>	<u>0.44</u>	0.36	**0.18**	<u>0.53</u>
CoDy-*local-gradient*	0.17	**0.42**	0.40	**0.41**	<u>0.16</u>	**0.54**

Table 5: Results on the fid_+ scores of the different explanation methods for explaining wrong and correct predictions. The best result for each experimental setting is **bold**, and the second best is <u>underlined</u>. If multiple explainers have the best score, they all are **bold**.

To allow for a more nuanced assessment, Figure 9 shows what fid_+ scores the explanation approaches achieve at different sparsity levels. The plots in this figure report the cumulative distribution of necessary explanations, i.e., counterfactual examples, over different $sparsity$ levels. Higher fid_+ values are considered better, while a low $sparsity$ is preferable. Across all datasets and settings, the T-GNNExplainer approach consistently performs worse than all combinations of CoDy with any selection strategy. This is evident because the fid_+ level of the T-GNNExplainer approach is below that of CoDy in the entire interval of $sparsity$ values from 0 to 1. Similarly, GreeDyCF with all but the *random* selection strategy outperforms T-GNNExplainer in most cases, with the exception of the experiment for wrong predictions on the UCI-Forums dataset, where T-GNNExplainer slightly outperforms GreeDyCF-*local-gradient* for very high sparsity values.

There is a substantial difference between the explanations for correct predictions and those for wrong predictions for the UCI-Messages and the Wikipedia datasets. For wrong predictions in both datasets, all explainers find necessary explanations in more than double the cases than for correct predictions. The reasons for this phenomenon may lie in what essentially differentiates correct from wrong predictions: For wrong predictions, the past information was misinterpreted by the link prediction function. For correct predictions, the past information was correctly interpreted. Thus, explaining wrong predictions with counterfactuals entails finding the past information that misleads the link prediction function. In contrast, finding a counterfactual example for a correct prediction requires removing information so that the prediction becomes incorrect, which could be challenging if nearly all past information is in line with the prediction. In contrast, finding misleading information could be an easier task. Regardless of the reasons for this difference in fid_+

	UCI-Messages		UCI-Forums		Wikipedia	
	Correct	Wrong	Correct	Wrong	Correct	Wrong
T-GNNExplainer	0.43	0.35	0.29	0.35	0.33	0.43
GreeDyCF-*random*	0.03	0.02	0.02	0.02	0.04	0.02
GreeDyCF-*temporal*	0.05	0.04	0.05	0.03	0.06	0.03
GreeDyCF-*spatio-temporal*	0.08	0.04	0.05	0.04	0.07	0.04
GreeDyCF-*local-gradient*	0.06	0.03	0.04	0.03	0.05	0.02
CoDy-*random*	0.08	0.05	0.09	0.06	0.07	0.07
CoDy-*temporal*	0.07	0.04	0.10	0.05	0.06	0.06
CoDy-*spatio-temporal*	0.06	0.04	0.08	0.04	0.07	0.05
CoDy-*local-gradient*	0.07	0.05	0.09	0.06	0.07	0.06

Table 6: Results on the *sparsity* score of the explanations in the different experimental settings.

levels, explaining wrong predictions is argued to be more informative than explaining correct predictions because explanations for correct predictions can be biased [3]. Explanations for correct predictions explain not only the model but also the phenomenon [3]. By comparison, wrong predictions solely explain the target model without introducing bias [3].

While CoDy with either the *local-gradient* or the *spatio-temporal* selection strategy performs best in most of the experimental settings, for correct predictions on the UCI-Forums dataset, it is outperformed by GreeDyCF-*spatio-temporal*. The reasons for this are unclear. It could be a statistical anomaly that appears on the specific sample that was drawn for the experiment. Section 7.2.5 shows that CoDy can outperform GreeDyCF-*spatio-temporal* for this setting when the number of search iterations is increased.

Overall, out of the compared methods, CoDy is the explainer best suited to address the objectives of maximizing the discovery of counterfactual examples while minimizing the complexity of the explanations. CoDy-*spatio-temporal* and CoDy-*local-gradient* consistently perform well in terms of finding necessary explanations with minimal sparsity scores. The largest advantage of CoDy can be observed in the setting of wrong predictions on the Wikipedia dataset, where CoDy-*spatio-temporal* outperforms the best GreeDyCF variant by 38% and T-GNNExplainer by 145% in terms of fid_+. However, GreeDyCF-*spatio-temporal* also demonstrates good performance in most experimental settings in terms of the objectives of maximizing discoveries and minimizing complexity.

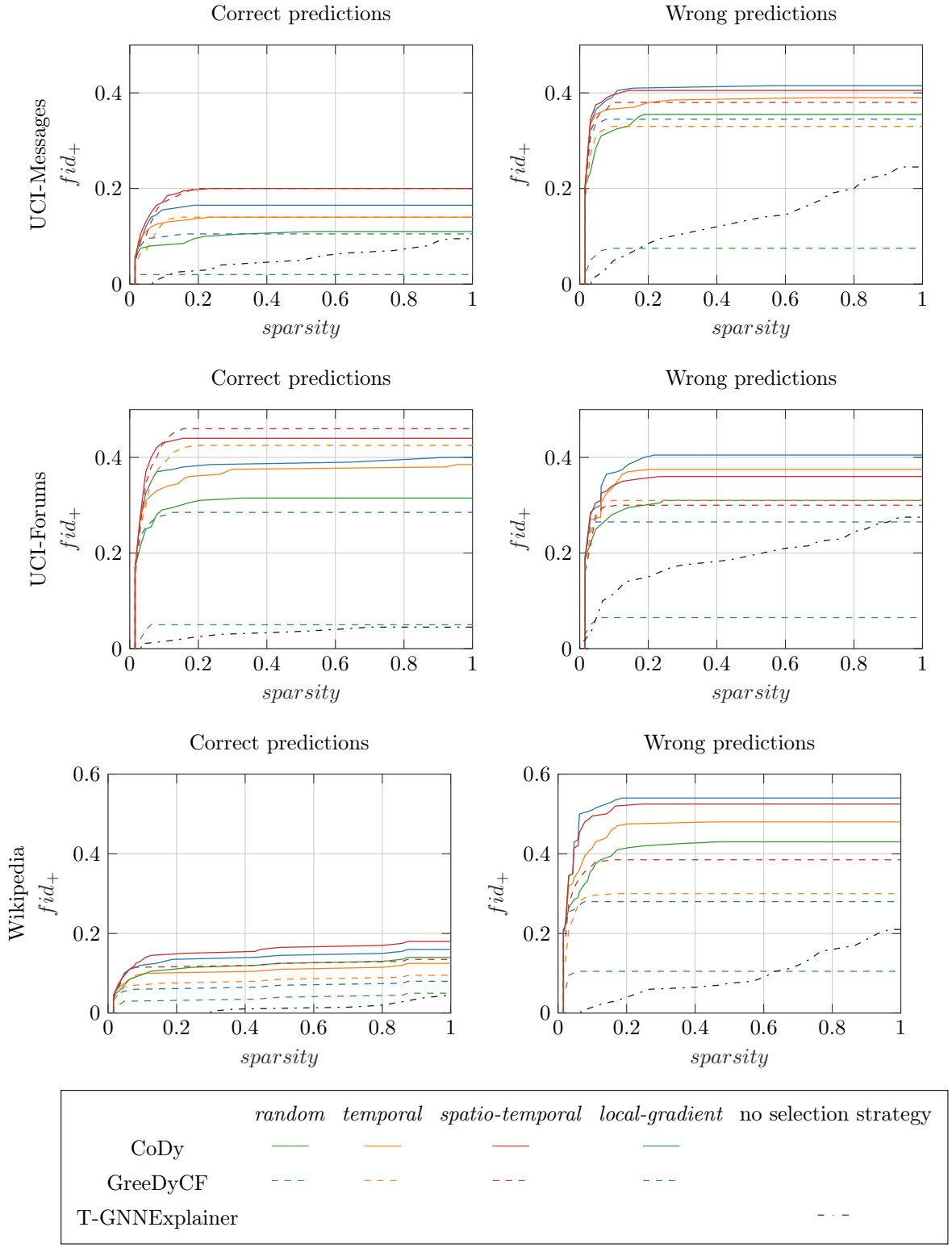


Figure 9: Results on the relation between fid_+ and $sparsity$ for the different experimental settings.

7.2.2 Sufficiency of Explanations

While the aim of the proposed explanation methods is to produce counterfactual explanations, which do not explicitly aim to achieve sufficiency, sufficiency is still an expressive trait of explanations [58]. Thus, this section assesses how the different approaches perform in terms of sufficiency. As discussed before, the fid_- metric is used to analyze the fraction of explanations that achieve sufficiency. For this metric, the explanations are interpreted as factual rather than counterfactual explanations. Table 7 shows how the different approaches perform in this metric. Since T-GNNExplainer is developed to provide factual explanations, it would be of no surprise if it outperformed the other methods in terms of fid_-. While this is the case for correct predictions on the UCI-Messages dataset, there are other explainers that perform best in the remaining settings. A potential explanation for this may lie in the implementation of T-GNNExplainer. For this evaluation, the original implementation of T-GNNExplainers was used, as proposed by its authors [67]. This implementation uses an approximation strategy for the results of the link prediction function. This approximation strategy may lead to predictions that differ substantially from the predictions without such approximation. Thus, this approximation can introduce errors into the predictions. This may partly explain why the fid_- score achieved by T-GNNExplainer is rather low across the experiments compared to the other explanation methods.

	UCI-Messages		UCI-Forums		Wikipedia	
	Correct	Wrong	Correct	Wrong	Correct	Wrong
T-GNNExplainer	**0.82**	0.83	0.57	0.72	<u>0.90</u>	0.69
GreeDyCF-*random*	0.34	0.97	0.28	**0.99**	0.57	**0.93**
GreeDyCF-*temporal*	0.56	**0.98**	0.61	0.98	0.77	0.90
GreeDyCF-*spatio-temporal*	0.70	0.96	0.64	0.95	0.82	0.88
GreeDyCF-*local-gradient*	0.67	**0.98**	0.64	**0.99**	0.72	0.90
CoDy-*random*	0.68	0.95	0.66	0.97	0.88	0.88
CoDy-*temporal*	0.69	0.96	0.66	0.98	0.88	0.88
CoDy-*spatio-temporal*	<u>0.72</u>	0.95	**0.69**	0.93	**0.91**	0.86
CoDy-*local-gradient*	0.70	0.96	<u>0.68</u>	0.95	0.88	<u>0.90</u>

Table 7: Results on the fid_- scores for the different experimental settings. The best result for each experimental setting is **bold**, and the second best is <u>underlined</u>. If multiple explainers have the best score, they all are **bold**.

Interestingly, there is an apparent difference between correct and wrong predictions for the performance of CoDy and GreeDyCF. The pairings of CoDy with the different selection strategies outperform their respective GreeDyCF counterparts by between 3.5% and 57.6% for correct predictions. However, the GreeDyCF approaches outperform the respective

configurations of CoDy by up to 5.4% for wrong predictions. Generally, the fid_- scores achieved by CoDy and GreeDyCF for wrong predictions are very high, reaching levels of more than 0.85 across all experimental settings. This underlines the proficiency of these explanation methods in explaining wrong predictions and suggests that counterfactual explanations are associated with a high sufficiency for wrong predictions.

Additionally, CoDy-*spatio-temporal* and CoDy-*local-gradient* mostly perform similar or better than T-GNNExplainer in terms of fid_+. The high fid_+ scores achieved by these approaches demonstrate that the explanations they produce are mostly sufficient for explaining predictions. This means that the explanations successfully capture information that is relevant to the prediction.

7.2.3 Convergent Analysis of Sufficiency and Necessity

The characterization score *char* integrates the perspectives of sufficiency and necessity. Table 8 shows the performance of the explainers along this metric. While CoDy-*local-gradient* performs best in explaining wrong predictions, CoDy-*spatio-temporal* performs best in explaining correct predictions. This convergent perspective is also portrayed in Figure 10, which shows the performance of the explainers in both the fid_+ and the fid_- scores. The plots show that CoDy-*local-gradient* and CoDy-*spatio-temporal* generally provide the best explanations in terms of necessity while also providing comparably good performance in terms of sufficiency. This highlights the importance of the fallback strategy (see Section 6.4.5) that provides explanations even if no counterfactual examples are found. The fallback is important because even though GreeDyCF-*spatio-temporal* and GreeDyCF-*spatio-temporal* find counterfactual examples only for between 20% and 54% of the explained predictions depending on the experimental setting, they provide sufficient explanations in between 68% and 96% of the explained predictions depending on the setting. This means that while not all explanations these approaches produce are counterfactual examples, the explanations still highlight important input information.

	UCI-Messages		UCI-Forums		Wikipedia	
	Correct	Wrong	Correct	Wrong	Correct	Wrong
T-GNNExplainer	0.17	0.39	0.08	0.40	0.09	0.34
GreeDyCF-*random*	0.04	0.14	0.08	0.12	0.09	0.19
GreeDyCF-*temporal*	0.22	0.49	0.50	0.47	0.17	0.45
GreeDyCF-*spatio-temporal*	<u>0.31</u>	0.54	<u>0.53</u>	0.46	0.23	0.54
GreeDyCF-*local-gradient*	0.18	0.51	0.39	0.42	0.14	0.43
CoDy-*random*	0.19	0.52	0.43	0.47	0.24	0.58
CoDy-*temporal*	0.23	0.55	0.49	<u>0.54</u>	0.22	0.62
CoDy-*spatio-temporal*	**0.31**	<u>0.57</u>	**0.54**	0.52	**0.30**	<u>0.65</u>
CoDy-*local-gradient*	0.27	**0.58**	0.50	**0.57**	<u>0.27</u>	**0.68**

Table 8: Results on the *char* scores for the different experimental settings. The best result for each experimental setting is **bold**, and the second best is <u>underlined</u>.

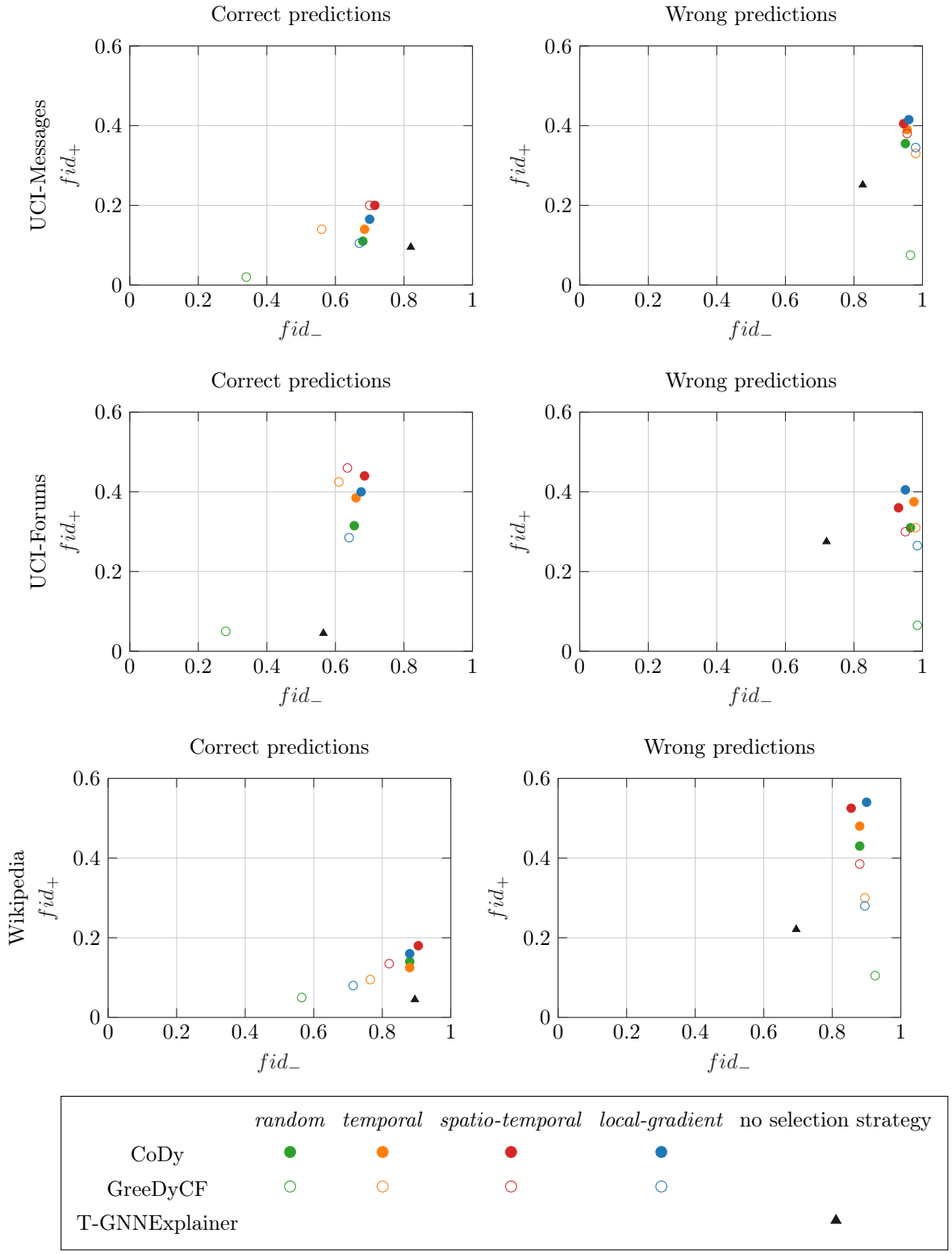


Figure 10: Results on the fid_+ score in relation to the fid_- score achieved by the explanation approaches.

7.2.4 Study of Runtime

Figure 11 shows the average time it takes the explanation approaches to explain a prediction. Across all datasets, GreeDyCF requires the least time on average. Within the variants of GreeDyCF, the *local-gradient* selection strategy entails a considerably longer time for each explanation. The reason for this difference is that the link prediction function is called a lot more often by GreeDyCF-*local-gradient* compared to the other variants of GreeDyCF. This can be seen in Table 9, which shows the average number of calls the evaluated explainers make to the link prediction function. The table shows that, in general, the link prediction function is called a lot more often by CoDy compared to GreeDyCF. This is not too surprising, as GreeDyCF concludes upon encountering a counterfactual example or when it cannot greedily advance in its search. On the other hand, CoDy only concludes after the predefined number of iterations or when the search tree is already fully explored.

Across all datasets and selection strategies, CoDy spends 97.85% of the explanation duration on calling the link prediction function, whereas GreeDyCF spends 99.83%, and T-GNNExplainer 82.72% on this. This shows that the main contributors to the runtime are the calls to the link prediction model. Therefore, using a faster link prediction model would also speed up explanation times. In contrast to the two explanation methods developed in this thesis, T-GNNExplainer spends considerably longer on the search procedure itself compared to calling the model.

As shown in Table 9, the number of calls to the model that are performed by each of the evaluated explanation approaches is relatively stable across the different datasets. However, comparing the average duration per explanation for CoDy and GreeDyCF in Figure 11 shows large differences between the datasets. CoDy and GreeDyCF take substantially more time per explanation on the Wikipedia dataset compared to the other two datasets. The reason for this difference lies in the longer time it takes to get a response from the targeted model when the Wikipedia dataset is used. A possible cause could lie in the fact that the Wikipedia dataset contains edge features, whereas the other two datasets do not. Another difference in the datasets that could account for the difference is that the Wikipedia dataset contains substantially more edges than the other datasets.

In general, GreeDyCF achieves relatively short explanation durations. This makes it the preferable method if fast explanations are a priority.

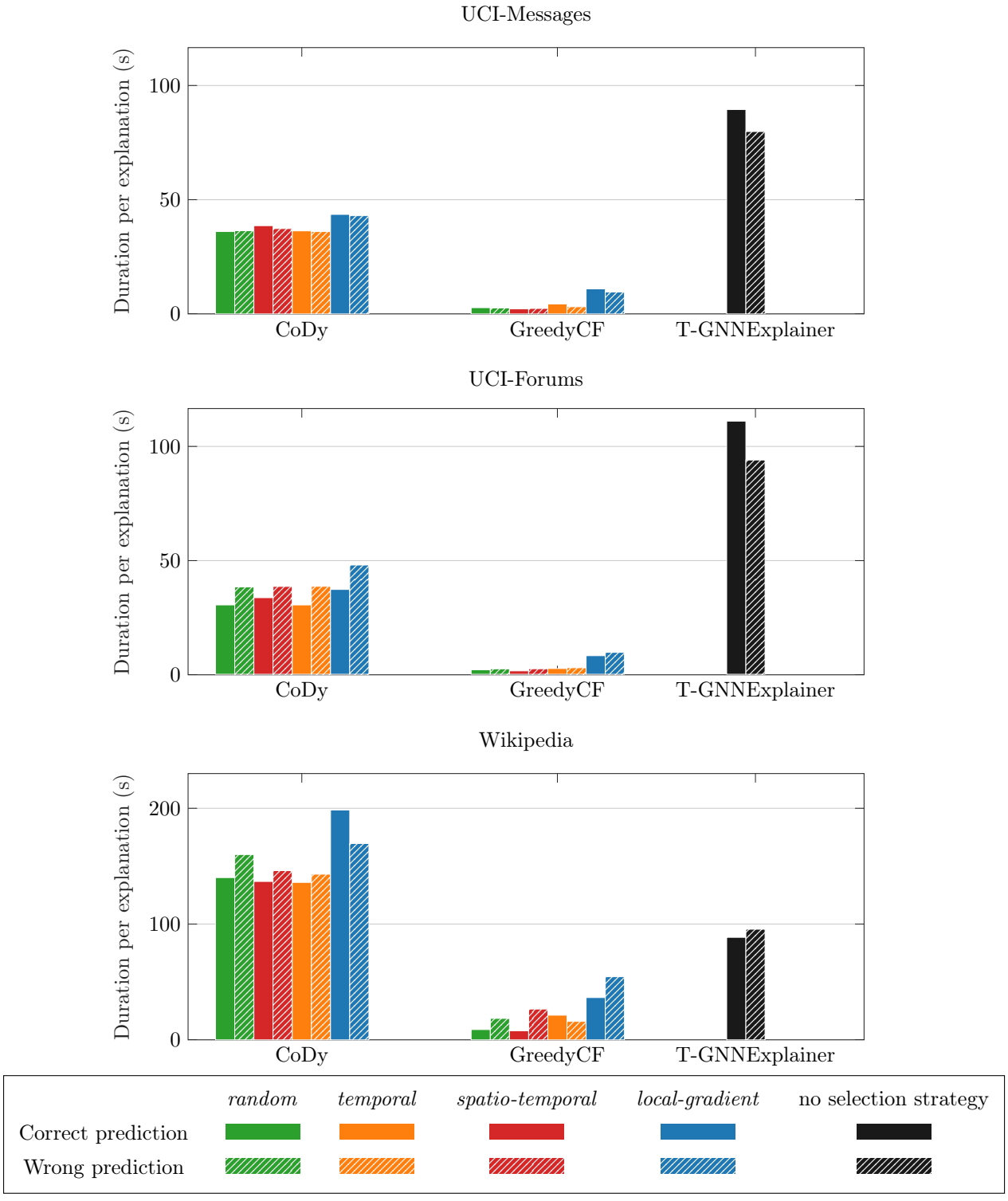


Figure 11: Results on the average duration of explaining a prediction for the different experimental settings.

	UCI-Messages		UCI-Forums		Wikipedia	
	Correct	Wrong	Correct	Wrong	Correct	Wrong
GreeDyCF-*random*	27.10	23.40	24.80	21.80	23.15	18.55
GreeDyCF-*temporal*	43.70	29.85	36.15	29.15	32.30	24.65
GreeDyCF-*spatio-temporal*	57.60	29.40	37.65	35.25	39.35	22.35
GreeDyCF-*local-gradient*	100.15	79.18	88.10	78.15	80.47	69.36
CoDy-*random*	287.91	246.83	261.13	257.23	284.52	237.16
CoDy-*temporal*	287.92	245.64	261.23	256.31	284.85	236.26
CoDy-*spatio-temporal*	287.95	245.44	260.95	256.44	284.98	236.26
CoDy-*local-gradient*	346.50	292.23	312.00	305.70	339.90	272.52

Table 9: Results on the average number of calls to the link prediction function performed by the explanation methods for the different experimental settings.

7.2.5 Study of Search Iterations

One of the configurable parameters of CoDy is the maximum number of search iterations it_{max}. Throughout the experiments, this number is set to 300. To investigate the performance impact of the number of search iterations, another experiment is conducted in which the maximum number of search iterations is raised to $it_{max} = 1200$. The experiment is conducted for correct prediction on the Wikipedia dataset because this is the only setting in which GreeDyCF-*spatio-temporal* outperforms all CoDy variants. Figure 12 shows the fid_+ score that is achieved in relation to the search iterations. This means it represents the fid_+ score that would be achieved if the maximum number of iterations was fixed to that of the number of iterations on the x-axis. The fid_+ scored for the GreeDyCF variants are independent of the iterations variable because this explainer does not have a configurable maximum number of search iterations. The figure demonstrates that all CoDy variants outperform their respective GreeDyCF counterparts at some number of maximum iterations. The number of iterations they take varies significantly, while for the *random* and *local-gradient* selection strategies a few iterations suffice to surpass their GreeDyCF counterparts, it takes almost 1000 iterations for CoDy-*spatio-temporal* and almost 1200 for CoDy-*temporal*.

Regardless of the selection strategy, CoDy is guaranteed to find the minimal counterfactual example when given an unlimited number of iterations. That is because with an unlimited number of iterations, CoDy traverses the entire search space, which means that it encounters all counterfactual examples that exist for any prediction instance. The fact that it takes CoDy-*spatio-temporal* and CoDy-*temporal* so many iterations to outperform their GreeDyCF counterparts suggests that the hyperparameters α and β that influence the balance between exploration and exploitation require optimization. This conclusion is evident because GreeDyCF outperforms these CoDy variants up to a high number of

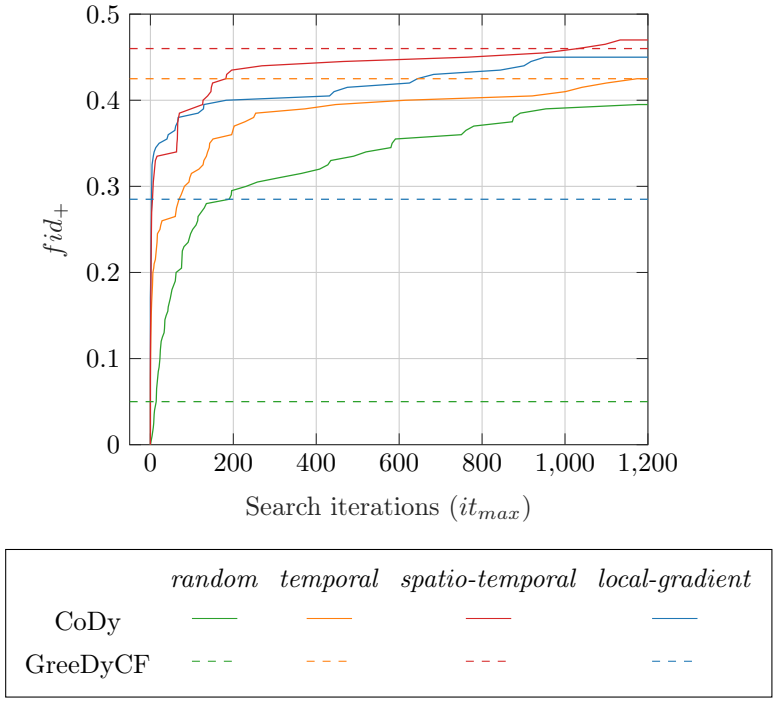


Figure 12: Results on the fid_+ scores achieved by the different explanation methods for correct predictions on the Wikipedia dataset. The fid_+ scores of the CoDy variants are depicted in relation to the number of search iterations.

iterations, which means that GreeDyCF explores parts of the search space that are not explored by CoDy. Since GreeDyCF constructs a substantially smaller partial search tree than CoDy, the CoDy variants should be able to construct a partial search tree that includes that of GreeDyCF. Since it evidently takes many iterations until it does so, the search may tend to either explore or exploit the partial search tree too much instead.

7.2.6 Study of Similarities in Explanations

The similarities of explanations are studied using the Jaccard similarity, which measures the overlap between explanations generated by different explainers. For the sake of simplicity, Figure 13 only shows a heatmap of the Jaccard similarities between the explanations for correct predictions on the Wikipedia dataset. However, the results are mostly alike across the different experimental settings. Thus, the results from Figure 13 serve as a representative example of the full results. The full results for all settings are presented in detail in Appendix A.1. The heatmap in Figure 13 shows the Jaccard similarity between the explanations of an explainer denoted in the column and those of another explainer denoted in the row.

Figure 13 shows that there is a very low similarity between the explanations produced by T-GNNExplainer and the other approaches. This result suggests that the factual explanations that T-GNNExplainer produces are substantially different from the counterfactual explanations produced by GreeDyCF and CoDy. Another contributing factor to these low similarity scores is that the explanations produced by T-GNNExplainer are generally much less sparse than those produced by the other explainers, as evident from Table 6.

When comparing the different GreeDyCF variants, a particularly high similarity exists between the explanations of GreeDyCF-*temporal* and GreeDyCF-*spatio-temporal*. Further, the explanations produced by GreeDyCF-*spatio-temporal* are most similar to those produced by CoDy with any selection strategy.

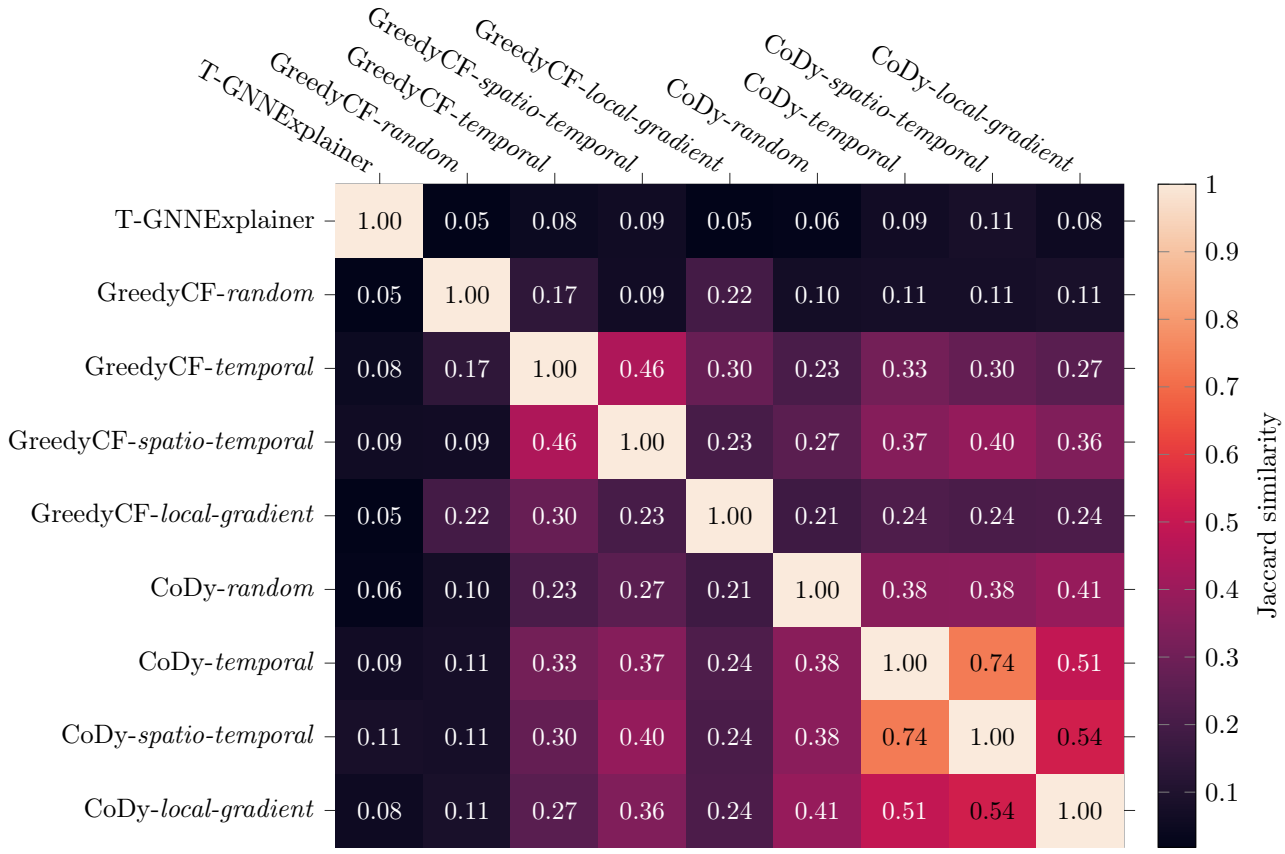


Figure 13: Results on the Jaccard similarity between explanations of correct predictions on the Wikipedia dataset.

The explanations produced by the different CoDy variants are generally more similar to each other than those of the GreeDyCF variants. A particularly high similarity exists between the explanations of CoDy-*temporal* and CoDy-*spatio-temporal*. Furthermore, CoDy-*temporal* and CoDy-*spatio-temporal* also achieve a high similarity with CoDy-*local-gradient*. This suggests that the flexibility that the search procedure of CoDy provides

allows CoDy to explore similar parts of the search space regardless of the selection strategy. It also highlights the similarities between the *temporal*, *spatio-temporal*, and *local-gradient* selection strategies, as these are associated with explanations that are more similar to each other compared to the *random* selection strategy.

7.2.7 Study of Selection Strategies

There exist substantial differences between the performance of GreeDyCF and CoDy depending on the selection strategy. This indicates that selecting events to include in the perturbation set based on these strategies influences the operations of the explanation approaches. If there were no effect of a strategy, the explainers would be assumed to perform similarly to the *random* selection strategy. However, the results for the metrics of fid_+ and fid_- show that for both CoDy and GreeDyCF, the *random* selection strategy is mostly outperformed by the other selection strategies. This suggests that the selection strategies aid the explainers in exploring relevant areas of the search space.

Taking a look at the results for the fid_+ scores achieved by the different GreeDyCF variants in Table 5 and Figure 9 clearly shows that the *spatio-temporal* selection strategy is performing best overall. The *temporal* selection strategy performs second best, while the *local-gradient* strategy comes in third. This shows that the spatial and temporal proximity of events has a large influence on the prediction of the target model for the investigated datasets. The same ranking of selection strategies also translates to the results on the characterization scores *char* (see Table 8), indicating the same findings when jointly analyzing sufficiency and necessity.

Shifting the view to the results for the fid_+ scores achieved by CoDy with the different selection strategies in Table 5 and Figure 9 shows that there is no clear best-performing selection strategy. While for explaining correct predictions, the *spatio-temporal* strategy performs best and the *local-gradient* performs second best, it is the other way around for explaining wrong predictions. Across all settings, CoDy-*temporal* generally performs third best in terms of fid_+. This ranking of selection strategies based on the necessity of explanations also transfers to a combined assessment of necessity and sufficiency using the characterization score *char* (see Table 8).

In general, the effects of the selection strategies on the explanatory performance are more pronounced for the GreeDyCF approach than for the CoDy approach. The reason for this could lie in the different methods for exploring the search space that these two explainers follow. CoDy is designed to exploit promising paths and to explore the search tree in more breadth if previously explored paths are not promising. Thus, if the selection strategy does not provide good initial selections, the explainer is incentivized to explore the search tree in other directions. In comparison, GreeDyCF remains on the path it has taken through

the search tree and only explores nodes that are ranked highly by the selection strategy. Thus, this approach gives the selection strategy more importance and is susceptible to concluding in local optima.

Across GreeDyCF and CoDy, the *spatio-temporal* selection strategy outperforms the *temporal* selection strategy in the majority of experiment settings. This relation evinces that spatial proximity is an important factor for the importance of past events to the prediction outcome of the targeted link prediction model. Thus, taking a combined spatio-temporal perspective on the past events serves as a particularly useful method for informing search-based counterfactual explanation methods. Furthermore, the *spatio-temporal* selection strategy has an advantage over the *local-gradient* strategy in terms of the time that is required for finding an explanation (see Figure 11).

8 Conclusion

In recent years, dynamic graphs and TGNNs have garnered increased attention because of their applicability to graphs that evolve over time, like in social networks [51] or financial transaction graphs [59]. Despite the opaque nature of most of the TGNN models that are used on dynamic graphs, the explainability of these black-box models remains mostly unexplored [67]. Specifically, counterfactual explanations have not been investigated for such models until now. Counterfactual explanations seek to uncover the elements of the input that are necessary for the explained prediction [58]. In comparison, the more common factual explanations seek to uncover the parts of the input that are sufficient for the explained prediction [58].

This thesis takes an event-based view of dynamic graphs. This means that a dynamic graph is fully described by an ordered list of timestamped events that add, remove, or update nodes and edges in the dynamic graph.

Taking a counterfactual perspective into explanations for the predictions of black-box models on dynamic graphs, this thesis proposes GreeDyCF and CoDy, two novel explanation methods that provide counterfactual explanations. To address the explanation problem, it is framed as a search task where the search space consists of all possible combinations of past events. Removing any combination of past events from the original input graph may change the prediction. If the removal changes the classification of the prediction, it makes that combination of past events a counterfactual example that is necessary to the explained prediction.

The search space is constrained in terms of its spatial and temporal extent, and it is structured into a search tree so that it facilitates the objectives of maximizing discoveries of counterfactual examples while minimizing the complexity of the explanation. GreeDyCF is proposed as a capable, low-complexity approach for searching for counterfactual explanations in this setting. It leverages a greedy search approach to find counterfactual examples within the search space quickly. As a more sophisticated approach, CoDy is introduced. CoDy adapts a search algorithm based on the Monte Carlo tree search algorithm to efficiently explore different potentially counterfactual perturbations to the input graph. In the absence of information on the importance of particular input events to the explained prediction, both search approaches leverage heuristic selection strategies to guide the search. These selection strategies are based on spatio-temporal proximity measures or local gradients.

Extensive evaluation reveals that CoDy outperforms GreeDyCF and a contemporary factual explanation method, providing counterfactual examples more frequently and with lower complexity while keeping the explanation time acceptable. GreeDyCF is shown to provide slightly worse explanations than CoDy but to perform best in terms of explana-

tion times. This positions GreeDyCF as a capable explanation approach for applications where timely explanations are essential. The experiments also uncover that the heuristic selection strategy has a large effect on explanatory performance. Using a selection strategy that considers spatio-temporal aspects and local gradients to select promising events for the explanation is preferable over random selection.

8.1 Limitations

Despite the contribution this work makes, there remain limitations that warrant consideration. These limitations exist on a conceptual and a practical level.

Conceptually, the very definition of the explanation problem undertaken in Section 5 presents a limitation. The problem definition limits counterfactual examples to perturbations that remove a set of past events from the input, but it does not regard other types of perturbations. While this limitation is essential for establishing a clear framework for the search process, it also imposes restrictions. Counterfactual examples do not have to be limited to the omission of information but could also encompass adding events to the input, changing the features associated with the nodes and edges in past events, or changing the timing of events. However, incorporating these dimensions would inevitably result in a substantial expansion of the search space, necessitating the development of novel methods to effectively navigate and address these more complex circumstances.

On a practical level, the applicability of CoDy faces two noteworthy constraints. Firstly, the relatively long explanation times and substantial resources required for explaining a single prediction confine the practical use of CoDy to cases where sufficient time is available. CoDy proves impractical for explaining large volumes of predictions within a reasonable timeframe. The main contributing factor to these temporal and resource requirements is the (in-)efficiency of the targeted model. Targeting better-optimized models would thus directly alleviate these concerns. Secondly, the fact that the proposed methods are not guaranteed to find a counterfactual example limits their usefulness. While this is partly mitigated by the fallback explanations, providing counterfactual examples for more of the explained instances would be preferable.

8.2 Future Work

Future work may address the limitations of this work. Further, it can explore avenues that build upon the presented findings and deepen the understanding of counterfactual explanations in the context of dynamic graphs.

The evaluation could be expanded to other target models. While TGN provides a framework that generalizes many approaches for TGNNs, extending the evaluation to other

target models provides an opportunity to assess whether the results gathered in this work generalize to other target models. Furthermore, assessing the explainers on a more diverse array of datasets could further establish the robustness and applicability of the developed approaches.

Improvements in the explanatory performance of CoDy could be achieved through systematic hyperparameter tuning, employing techniques such as grid search to optimize the balance between exploration and exploitation. Additionally, relaxing constraints on explanations, such as allowing the addition of events in perturbations instead of only removal, holds the potential for performance enhancement. However, this would come with the caveats discussed in Section 8.1.

Additionally, future work could apply and investigate GreeDyCF and CoDy to explain other tasks than future link prediction. For example, the explanation methods could be employed to explain the task of node classification. The main adaptation necessary for this application would be to update the Δ-function (see Equation 6.8) to quantify how much the certainty for the original node classification is shifted by a given perturbation set.

Lastly, another avenue for future research involves integrating counterfactual explanation methods with factual ones in the dynamic graph context. This has the potential of blending aspects of necessity and sufficiency into a unified framework. Hence, a hybrid approach allows for a more comprehensive understanding of the underlying graph dynamics.

A Appendix

A.1 Jaccard Similarities between explanation approaches

This section presents the Jaccard similarities achieved for the different experimental settings.

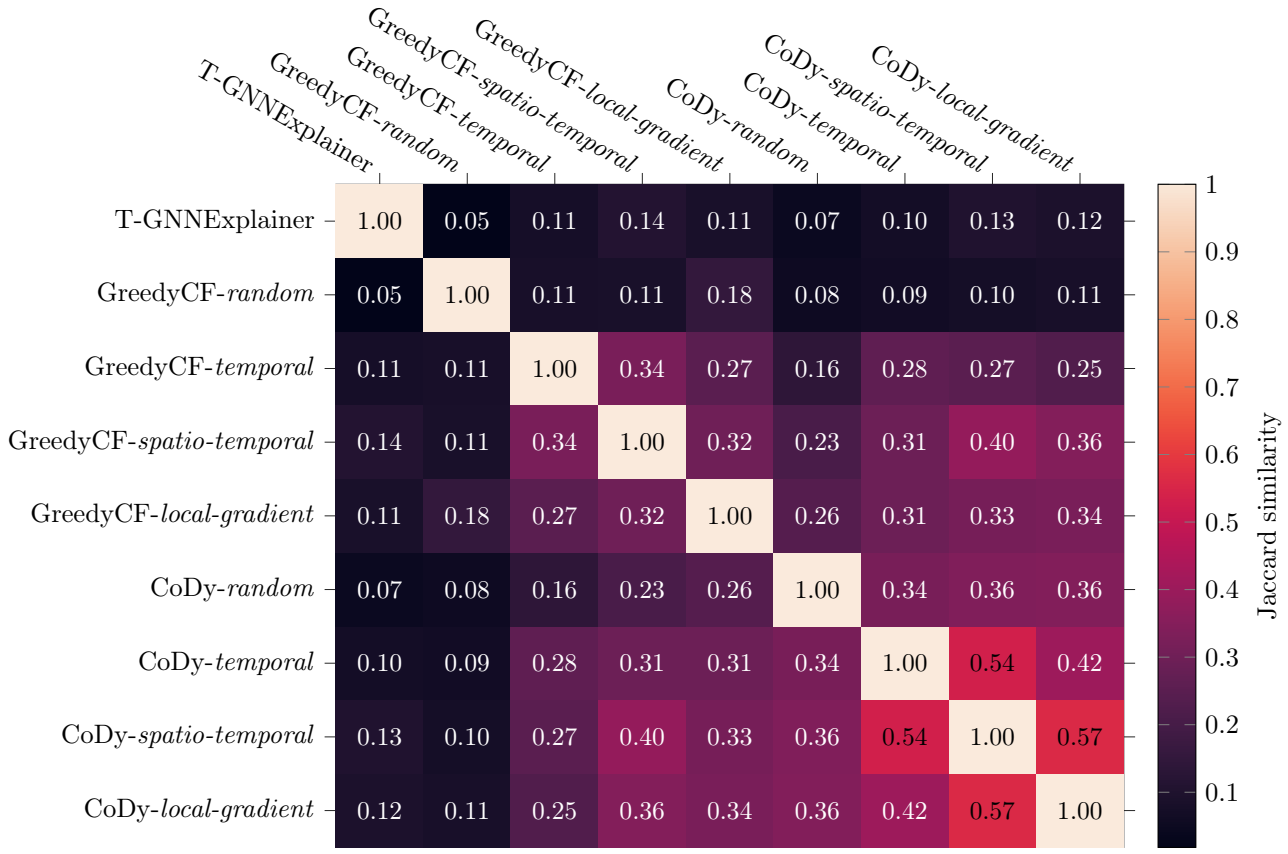


Figure 14: Results on the Jaccard similarity between explanations of correct predictions on the UCI-Messages dataset.

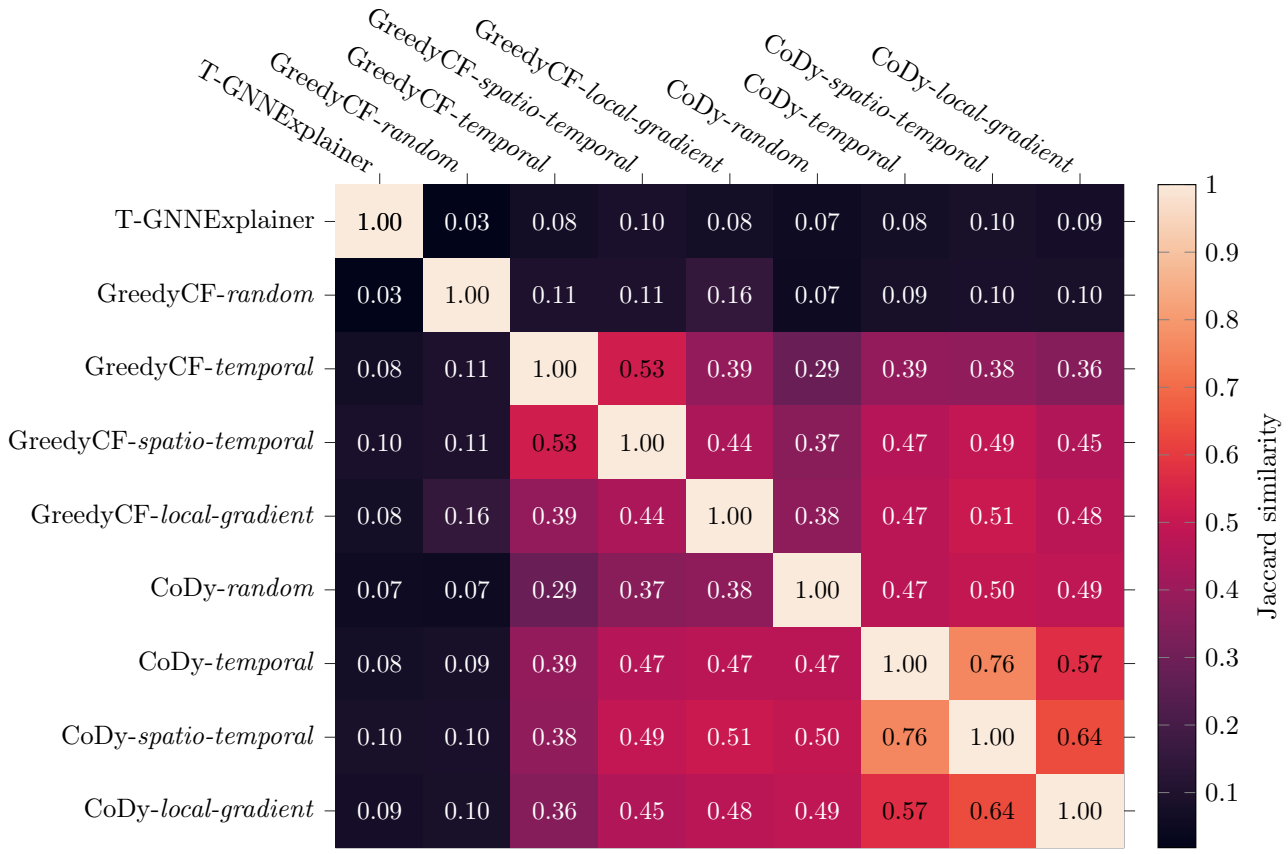


Figure 15: Results on the Jaccard similarity between explanations of wrong predictions on the UCI-Messages dataset.

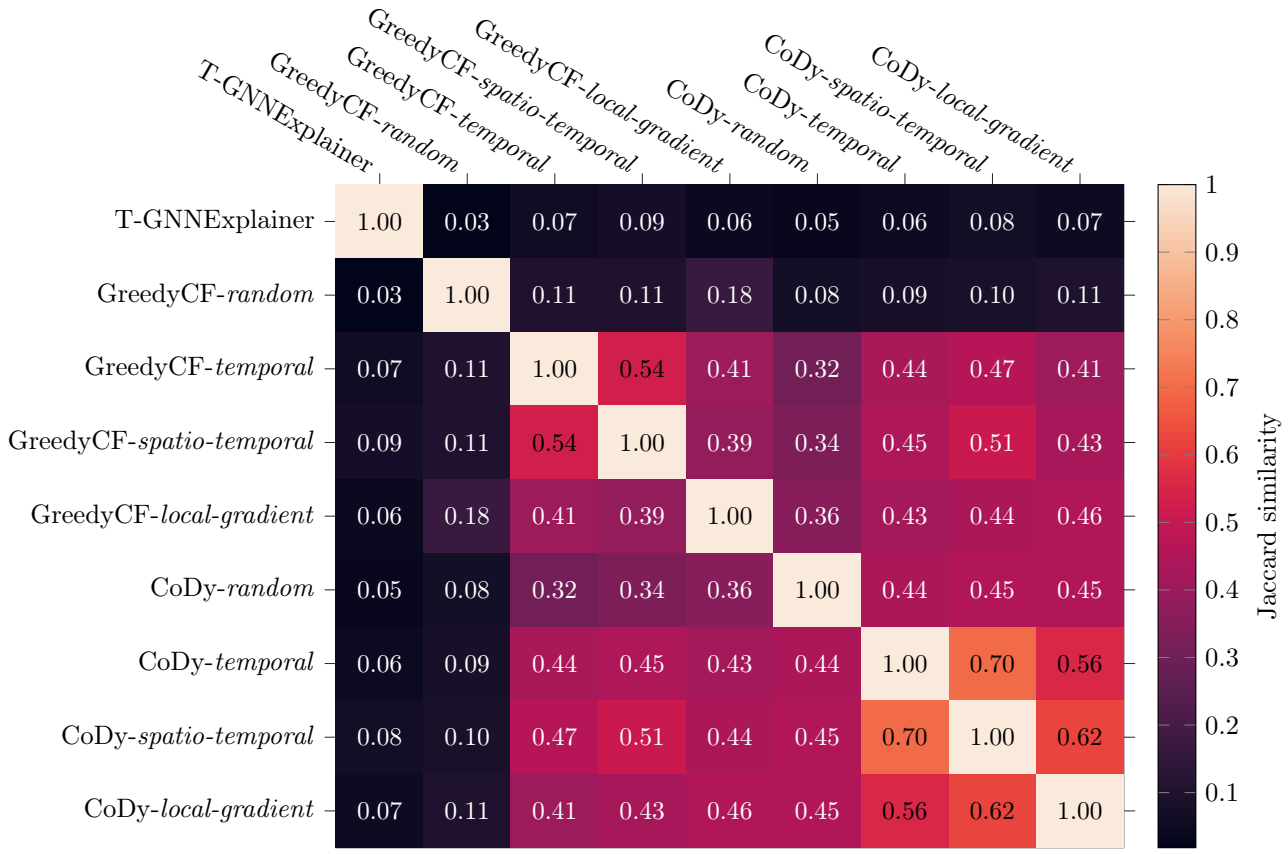


Figure 16: Results on the Jaccard similarity between explanations of correct predictions on the UCI-Forums dataset.

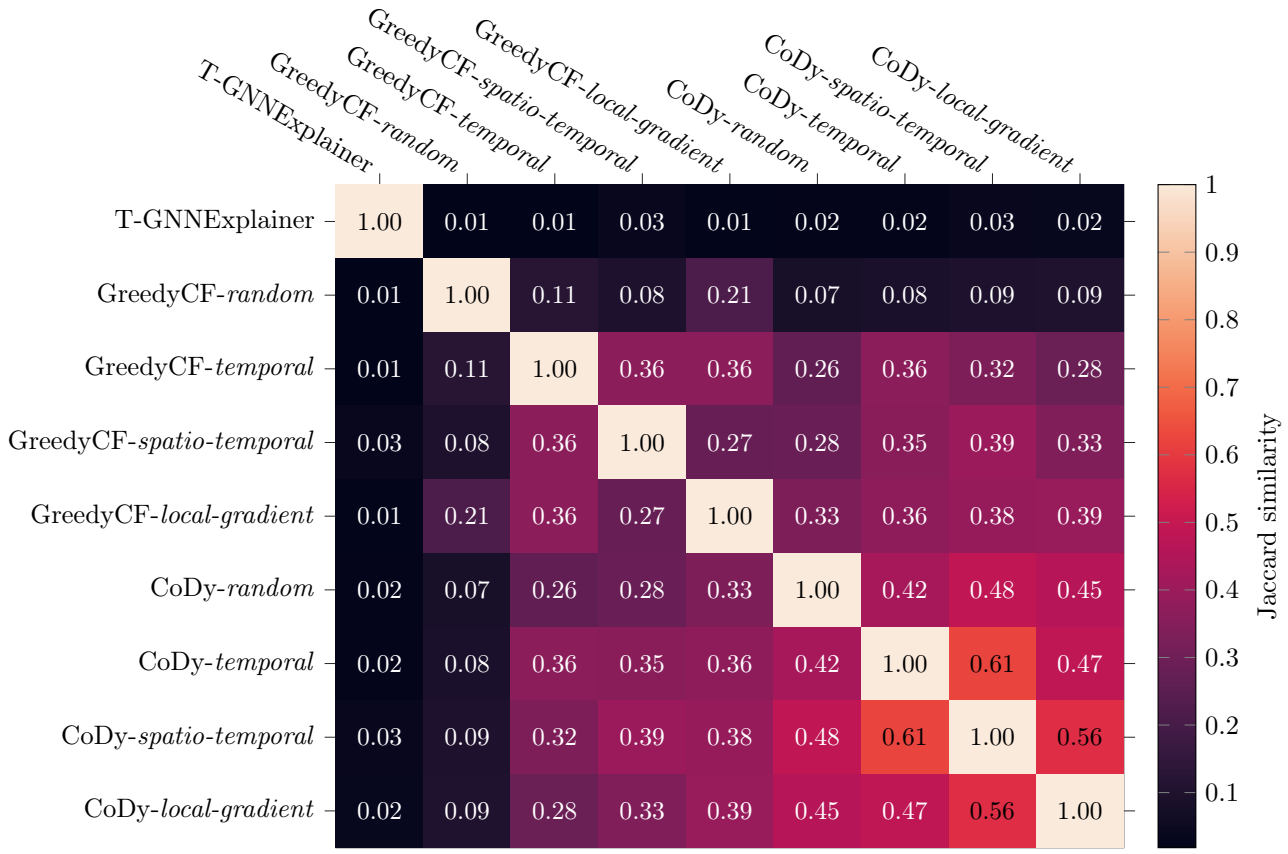


Figure 17: Results on the Jaccard similarity between explanations of wrong predictions on the UCI-Forums dataset.

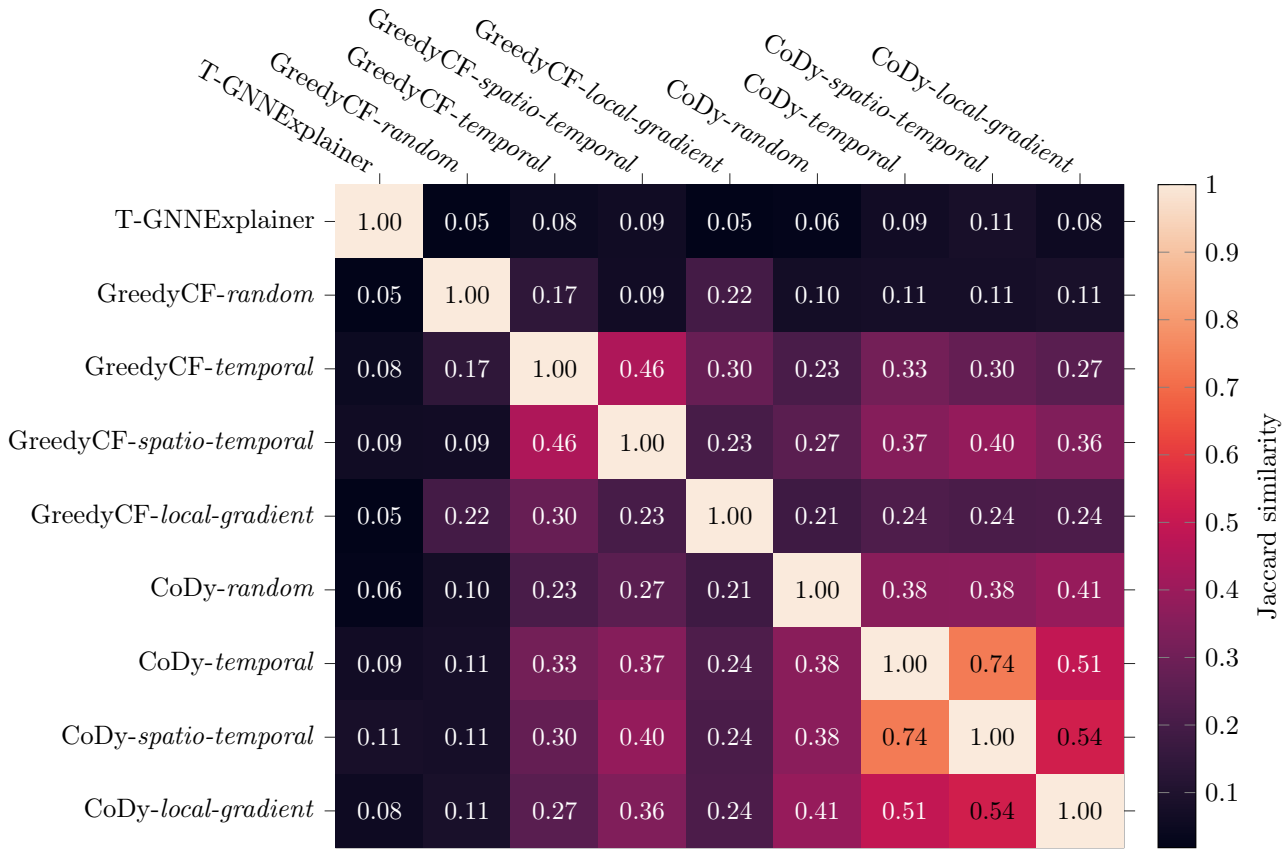


Figure 18: Results on the Jaccard similarity between explanations of correct predictions on the Wikipedia dataset.

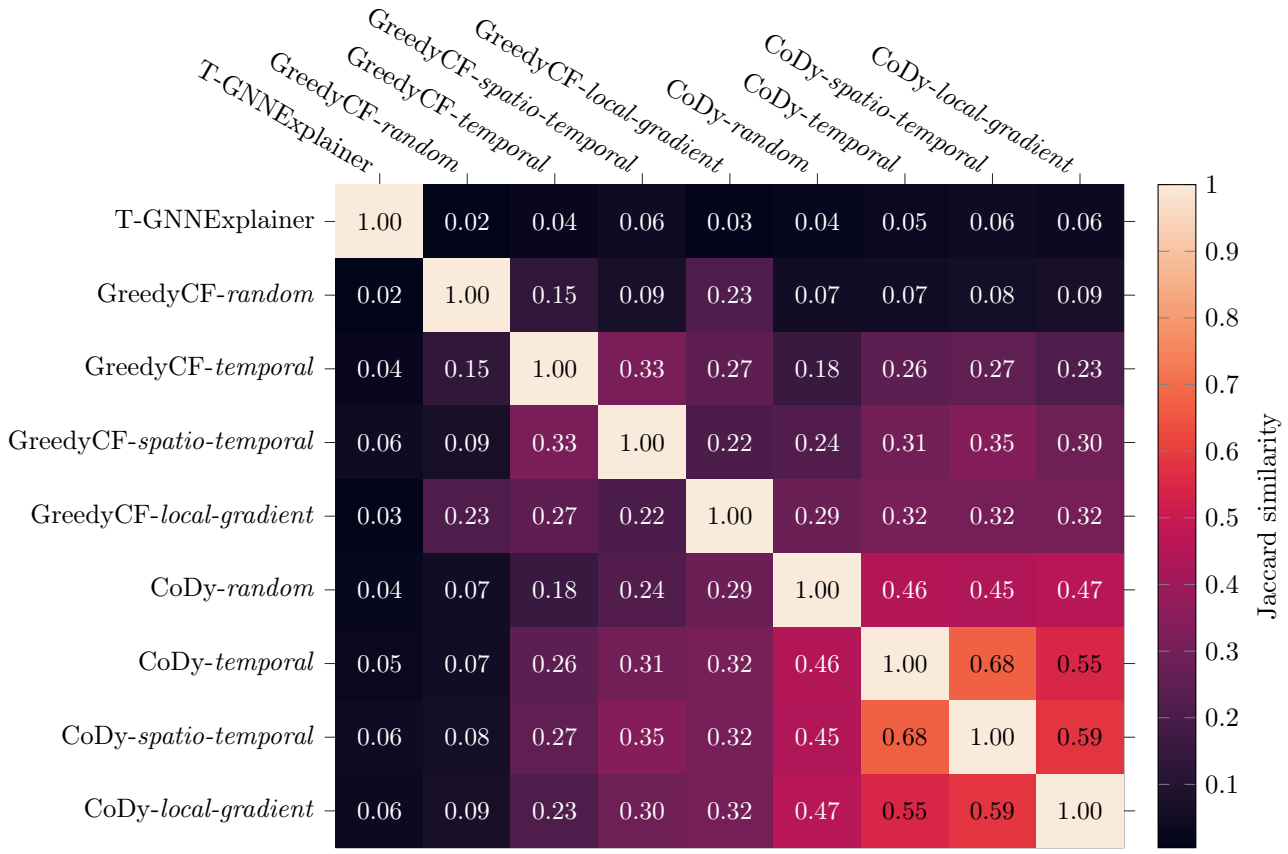


Figure 19: Results on the Jaccard similarity between explanations of wrong predictions on the Wikipedia dataset.

References

[1] Abrate, C., and Bonchi, F. Counterfactual Graphs for Explainable Classification of Brain Networks. In *ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (June 2021), arXiv. arXiv:2106.08640 [cs].

[2] Adadi, A., and Berrada, M. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). *IEEE Access 6* (2018), 52138–52160.

[3] Amara, K., Ying, R., Zhang, Z., Han, Z., Shan, Y., Brandes, U., Schemm, S., and Zhang, C. GraphFramEx: Towards Systematic Evaluation of Explainability Methods for Graph Neural Networks. In *The First Learning on Graphs Conference* (Oct. 2022), arXiv. arXiv:2206.09677 [cs].

[4] Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning 47*, 2 (May 2002), 235–256.

[5] Bajaj, M., Chu, L., Xue, Z. Y., Pei, J., Wang, L., Lam, P. C.-H., and Zhang, Y. Robust Counterfactual Explanations on Graph Neural Networks. In *35th Conference on Neural Information Processing System* (2021).

[6] Baldassarre, F., and Azizpour, H. Explainability Techniques for Graph Convolutional Networks. In *International Conference on Machine Learning* (May 2019).

[7] Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., and Herrera, F. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion 58* (June 2020), 82–115.

[8] Barros, C. D. T., Mendonça, M. R. F., Vieira, A. B., and Ziviani, A. A Survey on Embedding Dynamic Graphs. *ACM Computing Surveys 55*, 1 (Jan. 2023), 1–37.

[9] Bellman, R. Dynamic Programming. *Science 153*, 3731 (July 1966), 34–37. Publisher: American Association for the Advancement of Science.

[10] Bronstein, M. M., Bruna, J., Cohen, T., and Veličković, P. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges, May 2021. arXiv:2104.13478 [cs, stat].

[11] Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine 34*, 4 (July 2017), 18–42. arXiv:1611.08097 [cs].

[12] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems* (2020), vol. 33, Curran Associates, Inc., pp. 1877–1901.

[13] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games 4*, 1 (Mar. 2012), 1–43. Conference Name: IEEE Transactions on Computational Intelligence and AI in Games.

[14] Byrne, R. M. J. Counterfactuals in Explainable Artificial Intelligence (XAI): Evidence from Human Reasoning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence* (July 2019), pp. 6276–6282.

[15] Chen, J., and Ying, R. TempME: Towards the Explainability of Temporal Graph Neural Networks via Motif Discovery, Oct. 2023. arXiv:2310.19324 [cs].

[16] Dauparas, J., Anishchenko, I., Bennett, N., Bai, H., Ragotte, R. J., Milles, L. F., Wicky, B. I. M., Courbet, A., de Haas, R. J., Bethel, N., Leung, P. J. Y., Huddy, T. F., Pellock, S., Tischer, D., Chan, F., Koepnick, B., Nguyen, H., Kang, A., Sankaran, B., Bera, A. K., King, N. P., and Baker, D. Robust deep learning–based protein sequence design using ProteinMPNN. *Science 378*, 6615 (Oct. 2022), 49–56. Publisher: American Association for the Advancement of Science.

[17] Diestel, R. *Graph Theory*, vol. 173 of *Graduate Texts in Mathematics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2017.

[18] Duval, A., and Malliaros, F. D. GraphSVX: Shapley Value Explanations for Graph Neural Networks. In *Machine Learning and Knowledge Discovery in Databases. Research Track* (Cham, 2021), N. Oliver, F. Pérez-Cruz, S. Kramer, J. Read, and J. A. Lozano, Eds., Lecture Notes in Computer Science, Springer International Publishing, pp. 302–318.

[19] Fan, Y., Yao, Y., and Joe-Wong, C. GCN-SE: Attention as Explainability for Node Classification in Dynamic Graphs. In *2021 IEEE International Conference on Data Mining (ICDM)* (Dec. 2021), pp. 1060–1065. ISSN: 2374-8486.

[20] Goyal, P., and Ferrara, E. Graph Embedding Techniques, Applications, and Performance: A Survey. *Knowledge-Based Systems 151* (July 2018), 78–94. arXiv:1705.02801 [physics].

[21] Hamilton, W. L., Ying, R., and Leskovec, J. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin 40* (2017), 52 – 74. arXiv:1709.05584 [cs].

[22] He, W., Vu, M. N., Jiang, Z., and Thai, M. T. An Explainer for Temporal Graph Neural Networks. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference* (Rio de Janeiro, Brazil, Dec. 2022), IEEE, pp. 6384–6389.

[23] Huang, Q., Ren, H., and Leskovec, J. Few-shot Relational Reasoning via Connection Subgraph Pretraining. In *Advances in Neural Information Processing Systems* (May 2022).

[24] Huang, Q., Yamada, M., Tian, Y., Singh, D., and Chang, Y. GraphLIME: Local Interpretable Model Explanations for Graph Neural Networks. *IEEE Transactions on Knowledge and Data Engineering 35*, 7 (July 2023), 6968–6972. Conference Name: IEEE Transactions on Knowledge and Data Engineering.

[25] Ivanovs, M., Kadikis, R., and Ozols, K. Perturbation-based methods for explaining deep neural networks: A survey. *Pattern Recognition Letters 150* (Oct. 2021), 228–234.

[26] Jaccard, P. The Distribution of the Flora in the Alpine Zone.1. *New Phytologist 11*, 2 (1912), 37–50. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1469-8137.1912.tb05611.x.

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

[28] Kakkad, J., Jannu, J., Sharma, K., Aggarwal, C., and Medya, S. A Survey on Explainability of Graph Neural Networks. *IEEE Data Engineering Bulletin* (June 2023). arXiv:2306.01958 [cs].

[29] Kazemi, S. M., Goel, R., Jain, K., Kobyzev, I., Sethi, A., Forsyth, P., and Poupart, P. Representation Learning for Dynamic Graphs: A Survey. *Journal of Machine Learning Research 21* (2019), 70:1–70:73. arXiv:1905.11485 [cs, stat].

[30] Kipf, T. N., and Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations* (Feb. 2017). arXiv:1609.02907 [cs, stat].

[31] Kocsis, L., and Szepesvári, C. Bandit Based Monte-Carlo Planning. In *Machine Learning: ECML 2006* (Berlin, Heidelberg, 2006), J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds., Lecture Notes in Computer Science, Springer, pp. 282–293.

[32] Kumar, S., Zhang, X., and Leskovec, J. Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Anchorage AK USA, July 2019), ACM, pp. 1269–1278.

[33] Kunegis, J. KONECT – The Koblenz Network Collection. In *Proc. Int. Conf. on World Wide Web Companion* (2013), pp. 1343–1350.

[34] Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Alet, F., Ravuri, S., Ewalds, T., Eaton-Rosen, Z., Hu, W., Merose, A., Hoyer, S., Holland, G., Vinyals, O., Stott, J., Pritzel, A., Mohamed, S., and Battaglia, P. GraphCast: Learning skillful medium-range global weather forecasting, 2022. Publisher: arXiv Version Number: 2.

[35] Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Alet, F., Ravuri, S., Ewalds, T., Eaton-Rosen, Z., Hu, W., Merose, A., Hoyer, S., Holland, G., Vinyals, O., Stott, J., Pritzel, A., Mohamed, S., and Battaglia, P. GraphCast: Learning skillful medium-range global weather forecasting, Aug. 2023. arXiv:2212.12794 [physics].

[36] Liben-Nowell, D., and Kleinberg, J. The Link-Prediction Problem for Social Networks. *Journal of the American society for information science and technology 58*, 7 (2007), 1019–1031.

[37] Liu, Y., Zhang, X., and Xie, S. A Differential Geometric View and Explainability of GNN on Evolving Graphs. In *The Eleventh International Conference on Learning Representations (ICLR) 2023* (2023).

[38] Longa, A., Lachi, V., Santin, G., Bianchini, M., Lepri, B., Lio, P., scarselli, f., and Passerini, A. Graph Neural Networks for Temporal Graphs: State of the Art, Open Challenges, and Opportunities. *Transactions on Machine Learning Research* (2023).

[39] Lucic, A., ter Hoeve, M., Tolomei, G., de Rijke, M., and Silvestri, F. CF-GNNExplainer: Counterfactual Explanations for Graph Neural Networks. In *International Conference on Artificial Intelligence and Statistics* (Feb. 2022), arXiv. arXiv:2102.03322 [cs].

[40] Luo, D., Cheng, W., Xu, D., Yu, W., Zong, B., Chen, H., and Zhang, X. Parameterized Explainer for Graph Neural Network. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Nov. 2020), arXiv. arXiv:2011.04573 [cs].

[41] Ma, J., Guo, R., Mishra, S., Zhang, A., and Li, J. CLEAR: Generative Counterfactual Explanations on Graphs. In *36th Conference on Neural Information Processing Systems* (Nov. 2022). arXiv:2210.08443 [cs].

[42] Ma, Y., Guo, Z., Ren, Z., Zhao, E., Tang, J., and Yin, D. Streaming Graph Neural Networks. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Nov. 2018), arXiv. arXiv:1810.10627 [cs, stat].

[43] Makarov, I., Savchenko, A., Korovko, A., Sherstyuk, L., Severin, N., Mikheev, A., and Babaev, D. Temporal Graph Network Embedding with Causal Anonymous Walks Representations, Aug. 2021. arXiv:2108.08754 [cs].

[44] Manessi, F., Rozza, A., and Manzo, M. Dynamic Graph Convolutional Networks. *Pattern Recognition 97* (Jan. 2020), 107000. arXiv:1704.06199 [cs, stat].

[45] Parliament, E. Proposal for a regulation of the european parliament and of the council laying down harmonised rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts, 2021.

[46] Pennebaker, J. W., Francis, M. E., and Booth, R. J. Linguistic inquiry and word count: LIWC 2001. *Mahway: Lawrence Erlbaum Associates 71* (2001).

[47] Pope, P. E., Kolouri, S., Rostami, M., Martin, C. E., and Hoffmann, H. Explainability Methods for Graph Convolutional Neural Networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (Long Beach, CA, USA, June 2019), IEEE, pp. 10764–10773.

[48] Poursafaei, F., Huang, S., Pelrine, K., and Rabbany, R. Towards Better Evaluation for Dynamic Link Prediction. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track* (Sept. 2022), arXiv. arXiv:2207.10128 [cs].

[49] Prado-Romero, M. A., Prenkaj, B., Stilo, G., and Giannotti, F. A Survey on Graph Counterfactual Explanations: Definitions, Methods, Evaluation. *ACM Comput. Surv.* (Sept. 2023). arXiv:2210.12089 [cs].

[50] Quelhas, A. C., Rasga, C., and Johnson-Laird, P. N. The Relation Between Factual and Counterfactual Conditionals. *Cognitive Science 42*, 7 (2018), 2205–2228. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cogs.12663.

[51] Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., and Bronstein, M. Temporal Graph Networks for Deep Learning on Dynamic Graphs. In *Proceedings of the 37 th International Conference on Machine Learning* (Oct. 2020). arXiv:2006.10637 [cs, stat].

[52] Sankar, A., Wu, Y., Gou, L., Zhang, W., and Yang, H. DySAT: Deep Neural Representation Learning on Dynamic Graphs via Self-Attention Networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining* (Houston TX USA, Jan. 2020), ACM, pp. 519–527.

[53] Schnake, T., Eberle, O., Lederer, J., Nakajima, S., Schutt, K. T., Muller, K.-R., and Montavon, G. Higher-Order Explanations of Graph Neural Networks via Relevant Walks. *IEEE Transactions on Pattern Analysis and Machine Intelligence 44*, 11 (Nov. 2022), 7581–7596.

[54] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. Mastering the game of Go without human knowledge. *Nature 550*, 7676 (Oct. 2017), 354–359. Number: 7676 Publisher: Nature Publishing Group.

[55] Souza, A. H., Mesquita, D., Kaski, S., and Garg, V. Provably expressive temporal graph networks. In *Advances in Neural Information Processing Systems* (Sept. 2022), arXiv. arXiv:2209.15059 [cs].

[56] Stanley, R. P. *Enumerative Combinatorics*. Springer US, Boston, MA, 1986.

[57] Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Boston, MA, USA, June 2015), IEEE, pp. 1–9.

[58] Tan, J., Geng, S., Fu, Z., Ge, Y., Xu, S., Li, Y., and Zhang, Y. Learning and Evaluating Graph Neural Network Explanations based on Counterfactual and Factual Reasoning. In *Proceedings of the ACM Web Conference 2022* (Apr. 2022), pp. 1018–1027. arXiv:2202.08816 [cs].

[59] Trivedi, R., Farajtabar, M., Biswal, P., and Zha, H. DyRep: Learning Representations over Dynamic Graphs. In *International Conference on Learning Representations* (2019).

[60] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is All you Need. In *NIPS* (2017).

[61] Verma, S., Dickerson, J., and Hines, K. Counterfactual Explanations for Machine Learning: A Review. In *NeurIPS 2020 Workshop: ML Retrospectives, Surveys & Meta-Analyses (ML-RSA)* (2020).

[62] Vu, M. N., and Thai, M. T. PGM-Explainer: Probabilistic Graphical Model Explanations for Graph Neural Networks. In *Advances in Neural Information Processing Systems* (Oct. 2020), arXiv. arXiv:2010.05788 [cs].

[63] Vu, M. N., and Thai, M. T. On the Limit of Explaining Black-box Temporal Graph Neural Networks, Dec. 2022. arXiv:2212.00952 [cs].

[64] Wachter, S., Mittelstadt, B., and Russell, C. Counterfactual explanations without opening the black box: automated decisions and the GDPR. *Harvard Journal of Law and Technology 31*, 2 (2018), 841–887. Publisher: Harvard Law School.

[65] Wellawatte, G. P., Seshadri, A., and White, A. D. Model agnostic generation of counterfactual explanations for molecules. *Chemical Science 13*, 13 (Mar. 2022), 3697–3705. Publisher: The Royal Society of Chemistry.

[66] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems 32*, 1 (Jan. 2021), 4–24.

[67] Xia, W., Lai, M., Shan, C., Zhang, Y., Dai, X., Li, X., and Li, D. Explaining Temporal Graph Models Through An Explorer-Navigator Framework. In *The Eleventh International Conference on Learning Representations* (2023).

[68] Xie, J., Liu, Y., and Shen, Y. Explaining Dynamic Graph Neural Networks via Relevance Back-propagation, July 2022. arXiv:2207.11175 [cs].

[69] Xu, D., Ruan, C., Korpeoglu, E., Kumar, S., and Achan, K. Inductive Representation Learning on Temporal Graphs. In *International Conference on Learning Representations* (Feb. 2020). arXiv:2002.07962 [cs, stat].

[70] Ying, R., Bourgeois, D., You, J., Zitnik, M., and Leskovec, J. GNNExplainer: Generating Explanations for Graph Neural Networks. In *Advances in neural information processing systems* (Nov. 2019), arXiv. arXiv:1903.03894 [cs, stat].

[71] You, J., Du, T., and Leskovec, J. ROLAND: Graph Learning Framework for Dynamic Graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Aug. 2022), arXiv. arXiv:2208.07239 [cs].

[72] Yuan, H., Tang, J., Hu, X., and Ji, S. XGNN: Towards Model-Level Explanations of Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Aug. 2020), pp. 430–438. arXiv:2006.02587 [cs, stat].

[73] Yuan, H., Yu, H., Gui, S., and Ji, S. Explainability in Graph Neural Networks: A Taxonomic Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020). arXiv:2012.15445 [cs].

[74] Yuan, H., Yu, H., Wang, J., Li, K., and Ji, S. On Explainability of Graph Neural Networks via Subgraph Explorations. In *International Conference on Machine Learning* (May 2021), arXiv. arXiv:2102.05152 [cs].

[75] Zhang, S., Liu, Y., Shah, N., and Sun, Y. GStarX: Explaining Graph Neural Networks with Structure-Aware Cooperative Games. In *36th Conference on Neural Information Processing Systems* (2022).

[76] Zhao, L., Song, Y., Zhang, C., Liu, Y., Wang, P., Lin, T., Deng, M., and Li, H. T-GCN: A Temporal Graph ConvolutionalNetwork for Traffic Prediction. *IEEE Transactions on Intelligent Transportation Systems 21*, 9 (Sept. 2020), 3848–3858. arXiv:1811.05320 [cs, stat].

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

Assertion

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, December 12, 2023 Daniel Gomm