Titre: Title:	Contextual Embeddings and Graph Convolutional Networks for Concept Prerequisite Learning as a Strict Partial Order
Auteur: Author:	Jean-Charles Layoun
Date:	2024
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Layoun, J.-C. (2024). Contextual Embeddings and Graph Convolutional Networks for Concept Prerequisite Learning as a Strict Partial Order [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/58219/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/58219/
Directeurs de recherche: Advisors:	Michel C. Desmarais, & Amal Zouaq
Programme: Program:	Génie informatique

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Contextual Embeddings and Graph Convolutional Networks for Concept Prerequisite Learning as a Strict Partial Order

JEAN-CHARLES LAYOUN

Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

Génie informatique

Avril 2024

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

Contextual Embeddings and Graph Convolutional Networks for Concept Prerequisite Learning as a Strict Partial Order

présenté par **Jean-Charles LAYOUN**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

a été dûment accepté par le jury d'examen constitué de :

Samuel PIERRE, président
Michel DESMARAIS, membre et directeur de recherche
Amal ZOUAQ, membre et codirectrice de recherche
Daniel ALOISE, membre

DEDICATION

To my family, for their patience, love, and unwavering support.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor and co-supervisor, Professors Michel Desmarais and Amal Zouaq, for their guidance and valuable insights. I would also like to thank Professor Michel Desmarais for giving me the opportunity to teach as a lab assistant for the recommenders-system course. Moreover, I would like to thank both of my supervisors for giving me the chance to present our work at the ACM SAC conference in April 2024. Both of you were actively engaged and committed to our research, and through your rigor and work ethics, inspired me to strive harder and aim higher.

I am deeply grateful to all the funding agencies and governmental groups without which this project would not have been possible. Notably, the Social Sciences and Humanities Research Council of Canada supported this research by an INSIGHT grant.

I am also deeply grateful to all my friends and family who helped me throughout my research, specifically to Vincent Gagnon, who started his master's on this subject and paved the way for our research, to Ronan Sangouard who eagerly reviewed endless pages of mathematical related content without flinching, and to my sharp and brilliant sister, Dr. Marie Layoun, who never fails to answer my calls despite the 6 hours of time difference and the ocean between us. Lise, you have time and time again inspired me to be better. I hope you know that you are the most kind and generous human being I ever crossed path with, and even if times are rough for you right now, I have no doubt that you will find your way.

I am also deeply grateful for Professor Salim Salem's teachings. As an algebra professor, he gave me mathematical tools and reasoning that have been invaluable throughout my entire academic journey and will undoubtedly benefit my future research endeavors and scientific pursuits.

Finally, these last few lines are dedicated to my family. To my parents, who defied all logic with their selfless love and their support of my decision to fly across the Atlantic just so I could pursue my passions. To my sister, words are not enough and will never be enough to describe the privilege and joy that is to be your little brother. To Dr. Antoine Jabre and Dr. Jeanne-Marie Layoun, your incredible life stories inspire me to follow in your footsteps. I would be nowhere without the love and support of all of you, so thank you, and I love you.

RÉSUMÉ

L'Identification de relations de Prérequis entre les Concepts (IPC) joue un rôle crucial dans le milieu éducatif. L'objectif de l'IPC est de prédire les relations de prérequis entre différents concepts. Dans ce mémoire, nous présentons une nouvelle approche pour l'IPC utilisant les Sentence Transformers et les Réseaux de Neurones Relationnels à Convolution de Graphes (RNRCG). Cette approche crée des plongements de concepts à partir de définitions de concepts en utilisant des Transformers. Ces plongements sont ensuite utilisés comme entrée par un R-GCN, ainsi qu'une structure de graphe qui distingue les prérequis des non-prerequis.

Notre première contribution est liée à l'identification et à la résolution d'un problème méthodologique présent dans les études précédentes. Nous proposons un nouvel algorithme pour la création d'ensembles d'entrainement et de test qui réduit de 55% à 0% le pourcentage de relations dans l'ensemble de test déductibles par transitivité grâce aux relations dans l'ensemble d'entrainement. Le nouvel algorithme de division des données rend l'IPC plus difficile à résoudre, mais aussi plus conforme à la réalité de la tâche, car il exclut les inférences simples qui utilisent la propriété de transitivité des relations de prérequis.

Notre deuxième contribution concerne notre méthode de résolution de l'IPC. Notre RNRCG est optimisé simultanément afin de résoudre la tâche d'IPC et la tâche de classification du domaine des concepts. Ceci améliore la généralisation de l'identification des prérequis pour des domaines non vus dans l'entrainement. Nos expériences sur le jeu de données d'AL-CPL montrent l'efficacité de notre approche pour l'IPC intradomaine et interdomaine. En effet, nos résultats surpassent les méthodes de l'état de l'art sur ce jeu de données.

ABSTRACT

Concept prerequisite learning (CPL) plays a crucial role in education. The objective of CPL is to predict prerequisite relations between different concepts. In this thesis, we present a new approach for CPL using Sentence Transformers and Relational Graph Convolutional Networks (R-GCNs). This approach creates concept embeddings from single-sentence definitions extracted from Wikipedia that are fed to a Transformer. These embeddings are then used as an input feature matrix for the R-GCN, in addition to a graph structure that distinguishes prerequisites and non-prerequisites as distinct link types.

For our first contribution, we identified and solved a methodological problem present in previous studies. We introduced a novel algorithm for creating the training and test sets that reduced the percentage of relations in the test set that can be inferred by transitivity through the relations in the training set from 55% to 0%. The new data splitting algorithm makes CPL harder to solve, but also more in line with the reality of the task, since it excludes simple inferences that use the transitivity property of prerequisite relations.

Our second contribution is the model we developed to solve CPL. Our model is based on R-GCN layers that are optimized simultaneously on CPL and concept domain classification to enhance prerequisite prediction generalization for unseen domains. Extensive experiments on the AL-CPL dataset show the effectiveness of our approach for the in-domain and cross-domain settings, as it outperforms the State-Of-The-Art (SOTA) method on this dataset.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LISTE OF SYMBOLS AND ABBREVIATIONS

AL-CPL	Active Learning for Concept Prerequisite Learning
BERT	Bidirectional Encoder Representations from Transformers
CBOW	Continuous Bag of Words
CNN	Convolutional Neural Network
CPL	Concept Prerequisite Learning
DAG	Directed Acyclic Graph
GCN	Graph Convolutional Networks
IPC	Identification de relations de Prérequis entre les Concepts
LDA	Latent Dirichlet Allocation
LI	Loop Invariant
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multi-Layer Perceptron
MRR	Mean Reciprocal Rank
MOOC	Massive Open Online Course
MultiNLI	Multi-Genre Natural Language Inference
PnP	Prerequisite and non-Prerequisite
R-GCN	Relational Graph Convolutional Networks
SGD	Stochastic Gradient Descent
SNLI	Stanford Natural Language Inference
SOTA	State-Of-The-Art
VGAE	Variational Graph Auto-Encoder

CHAPTER 1 INTRODUCTION

1.1 General Context

Organizing educational content is challenging, especially when determining the relationships between concepts. Properly structured curricula enhance student learning, as they help form internal conceptual structures (Rakesh Agrawal et al., 2015). By systematically mapping course material, educators and researchers can gain valuable insights into the hierarchical structure of knowledge domains, therefore creating more efficient intelligent tutoring systems (Aleven & Koedinger, 2002).

This thesis focuses on the task of Concept Prerequisite Learning (CPL), aiming to identify if one concept is a prerequisite for another. Concept A is considered a prerequisite for concept B if and only if understanding concept B necessitates understanding concept A first. Here a concept refers to an abstract idea represented by one or more words. Therefore, automating the solving of CPL through algorithms could create a system for mapping course material that enhances student learning (Rakesh Agrawal et al., 2015). This could benefit teachers and students, especially in Massive Open Online Courses (MOOCs), where a large amount of educational resources are presented to the student by the teacher (Liang, Ye, Wang, et al., 2018).

However, solving CPL can be challenging in certain settings that are more representative of real-world scenarios than others. This thesis aims to solve CPL in such settings, despite the difficulties that it may cause.

1.2 CPL: a Task of Ordering Concepts

Students are presented with new knowledge sequentially according to a specific curriculum. One question arises: in which order must we present knowledge in order to maximize student learning? The hypothesis behind the motivation of CPL is that prior knowledge has a significant influence on the acquisition of new knowledge (Rittle-Johnson et al., 2009). The term "prior knowledge" with respect to a learning task is defined as knowledge that is not learned from the task itself but that increases the success of the given learning task. Scandura (1966) examined the impact of prior knowledge on task success by providing prerequisite information to one group and withholding it from another, then comparing both groups' performance on the same task. Others rely on table of contents, a consensus of domain experts, and logic to determine the order

in which concepts should be presented to students (Wang et al., 2016). Additionally, researchers have explored the automation of CPL using algorithms. This process entails categorizing concept pairs as either prerequisite or non-prerequisite in a binary fashion:

Table 1.1: An example of ordered concept pairs with respect to the prerequisite relation.

Concept	**Prerequisite**	**Is a Prerequisite Relation**
Right Angle	Angle	1
Right Triangle	Right Angle	1
Right Triangle	Angle	1
Triangle	Rectangle	0
Angle	Right Triangle	0

If "Angle" is a prerequisite to "Right Angle", then the concept pair will be labeled 1 as it is a prerequisite relation. Otherwise, if Rectangle is not a prerequisite to Triangle, then the concept pair will be labeled as 0 as it is a non-prerequisite relation (refer to Table 1.1).

In this thesis, we follow this binary formalization of CPL[1].

1.3 Research Questions

The objective of this thesis is to design a Machine Learning (ML) model to solve CPL based on concept embeddings. This method must generalize well to concepts from unseen domains and must be robust and applicable to the real-world scenario of the CPL task in education. This objective can be divided into two sub-objectives. These two sub-objectives constitute the two main research questions of this thesis, and thus the two main contributions of this thesis.

RQ1: How to leverage relational information with linguistic information for the CPL task using pretrained embeddings and Graph Neural Networks?

[1] However, this binary reasoning does not entirely grasp the complexity of the prerequisite relations between concepts. Indeed, not all prior knowledge is equally crucial for a given concept. To illustrate, both "Point" and "Right Angle" are prerequisites to "Right Triangle", yet understanding what a "Right Angle" is crucial to understand what a "Right Triangle" is. The binary representation of concept pairs fails to capture this nuance.

RQ2: Does the random split in the k-fold cross-validation method used in previous work create data splits where information leaks from the training set to the test set, making CPL an easier task to solve?

First, we investigate the potential for information leakage in data splits used in prior studies, which could inadvertently simplify the CPL task. Then we test multiple methods to assess if indeed the new split method renders CPL harder to solve. Finally, we test our own graph-based ML method for CPL that uses embeddings generated from text. This method outperforms the State-Of-The-Art (SOTA) methods.

Through these focused objectives and corresponding tasks, this thesis aims to advance the field of CPL by addressing current methodological shortcomings and harnessing the power of textual semantic representations to improve educational technologies.

1.4 Thesis Outline

CHAPTER 2 presents the background to our work in the field of CPL, including graph and set theory, semantic representations in natural language processing, and link prediction with graph convolutional networks. Additionally, it reviews related work and identifies the shortcomings of the State-Of-The-Art (SOTA) in CPL.

CHAPTER 3, titled “Concept Prerequisite Relations as a Strict Partial Order”, presents the statistics and intricacies of the AL-CPL dataset while discussing different possible graph representations. We then state the founding hypotheses of the AL-CPL, draw conclusions regarding the dataset and test if the AL-CPL satisfies all the mathematical properties outlined in the previous mathematical preliminaries section. CHAPTER 3 ends by identifying a methodological flaw in the splitting strategy employed by previous work and then elaborates on the solution we devised to tackle this flaw which is the Graph Split Algorithm.

Finally, in CHAPTER 4, we elaborate on our solution for solving CPL using an R-GCN and a Sentence Transformer. First, we describe the methodology of our solution. Then, we explain the experimental setup: we keep the same setup as previous work for comparison purposes, and we elaborate a new experimental setup that considers the methodological flaw present in previous methodologies described in 2.1. We conclude CHAPTER 4 by showcasing the results of our experiments on the AL-CPL dataset.

CHAPTER 2 Background and Related Work

The background and related work is divided into four sections. The first section deals with the theoretical background of concept prerequisite relations (Section 2.1). It forms the basis of our work. Then, Section 2.2 provides the background related to semantic representations of words in natural language processing that is necessary to understand the approach taken in this thesis. Consequently, Section 2.3 provides background on graph-based ML methods, which is also essential to understand the approach taken in this thesis. Finally, this chapter concludes with a review of previous research related to CPL and a discussion of its limitations (Section 2.4).

2.1 Mathematical Preliminaries for Concept Prerequisite Relations as a Strict Partial Order

The purpose of this section is to introduce the mathematical properties associated with concept prerequisite relations seen as a strict partial order. It presents the theoretical background necessary to understand the work in this thesis.

Indeed, our work in this section is heavily inspired by books on Graph Theory (*Graph Theory*, 2008) and Set Theory (*Set Theory*, 2003). In addition, previous work has already delved into the mathematical properties of concept prerequisite relations (Liang, Ye, Zhao, et al., 2018). However, for the sake of brevity and efficiency, Liang, Ye, Zhao, et al. (2018) have done so using advanced concepts in mathematics that are less accessible to the general public. Therefore, the definitions presented in this section are taken from books in the respective fields. However, the authors rewrote the propositions and proofs by using simple concepts and simple reasoning to make them more accessible for the purpose of this thesis.

Definition 1 (Prerequisite Relation between two Concepts). *Given two concepts A, and B, concept A is a prerequisite for concept B if and only if in order to understand B, A must be understood first.*

In the definition above, the word '*first*' is underlined because it renders the notion of order inherent to prerequisite relations between concepts.

Definition 2 (Homogeneous Binary Relation on a Set). *Given a set S, a homogeneous binary relation on set S defines a subset* $R \subseteq S \times S$ *where* $(x, y) \in R$ *means "x is R-related to y" and is denoted by* xRy.

In the following, the prerequisite relation between concepts A and B will be noted A→B.

Definition 3 (Strict Partial Order). *A strict partial order R on set S is a homogeneous binary relation on S that follows the following properties:*

1. Irreflexivity: $\forall \boldsymbol{x} \in \boldsymbol{S}, \neg(\boldsymbol{xRx})$.
2. Asymmetry: $\forall(\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{S^2}, \boldsymbol{xRy} \Rightarrow \neg \boldsymbol{yRx}$.
3. Transitivity: $\forall(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \in \boldsymbol{S^3}, \boldsymbol{xRy} \wedge \boldsymbol{yRz} \Rightarrow \boldsymbol{xRz}$.

Examples:

- Less than relation $<$ and greater than $>$ relation on real numbers $\mathbb{R}$ are strict partial orders.
- The set inclusion relation $\subset$ is a strict partial order on the set of subsets of a given set.
- The relation $\boldsymbol{R} = \{(\boldsymbol{a}, \boldsymbol{b})\}$ is a strict partial order on the lower-case letters of the English alphabet $\{\boldsymbol{a}, \boldsymbol{b}, \ldots, \boldsymbol{y}, \boldsymbol{z}\}$.

Proposition 1. *An irreflexive and transitive homogeneous binary relation is asymmetric.*

Proof:

We will proceed by contradiction:

Let R be a strict partial order on S. Assume that R is irreflexive and transitive without being asymmetric. This means that $\exists(x, y) \in S^2, xRy \wedge yRx$. By transitivity we deduce xRx. However, R is irreflexive, leading to a contradiction.

Q.E.D.

Proposition 1 implies that in order to be a strict partial order, a homogeneous binary relation only has to be irreflexive and transitive. We will use **Proposition 1** to prove that prerequisite relations define a strict partial order on concepts.

Proposition 2. *Prerequisite relations define a strict partial order on concepts.*

Proof:

According to **Proposition 1,** we only need to prove the irreflexivity and transitivity properties to prove that prerequisite relations define a strict partial order on concepts. Indeed, prerequisite relations are irreflexive and transitive by **Definition 1**:

- **Irreflexivity:** By definition, prerequisite relations are irreflexive because it is impossible to understand a concept before understanding the same concept.

  Indeed, let's proceed by contradiction. Let's suppose that there exists a concept A that can be understood for which the sentence '*A must be understood first in order to understand A'* is true. In that case, we fall into a contradiction: it is impossible to understand A because it is a prerequisite for itself.

- **Transitivity:** Let's suppose concept A, B and C are understandable concepts. If A is a prerequisite to B and B a prerequisite to C, then C can't be understood without understanding B first and B cannot be understood without understanding A first. Therefore, C cannot be understood without understanding A first ⇒ A is a prerequisite to C.

Q.E.D.

Definition 4 (Strict Total Order). *A strict total order is a strict partial order where any two distinct elements of S are comparable:* $\forall(x, y) \in S^2, x \neq y \Rightarrow xRy \vee yRx$.

Before moving on to graph-related definitions and propositions, it is important to explain why using graphs to represent prerequisite relations is relevant to our work and our field. **Graphs** can be applied to various fields and are easily interpretable. They model pairwise relations between vertices by connecting them with edges, providing a visual aid that is easily comprehensible to humans. In the field of education, graph representations of concept prerequisite relations are very useful because they clearly convey a mind map for the teachers and the students to follow. It improves the understanding of the logical connections between the concepts that are being introduced in a course (Rakesh Agrawal et al., 2015).

Finally, this thesis proposes an approach that heavily relies on graph representations for its algorithms and explanations. Therefore, the following paragraphs will establish the relationship between CPL and graph theory.

Definition 5 (Directed Graphs). *A directed graph* G *is comprised of a set of vertices* V *also called nodes and a set of edges* $E \subseteq V \times V$ *also called directed edges.*

Definition 6 (Directed Path). *For a directed graph G, a sequence of directed edges $(e_1, \dots, e_{n-1})$ where $n \in \mathbb{N}^*$, $e_i = (v_i, v_{i+1}) \in E$, and $\forall i \in [\![1, \mathrm{n}]\!], v_i \in V$ is called a directed path.*

Definition 7 (Directed Trail). *A directed trail in a directed graph G is a directed path where all edges are distinct: $\forall (i, j) \in [\![1, \mathrm{n}]\!]^2$, $e_i = (v_i, v_{i+1}) \in E$ and if $\mathrm{j} \neq i$ then $e_i \neq e_j$.*

Definition 8 (Directed Cycle). *A directed cycle in a directed graph G is a directed trail in which the first and last node are the same.*

Definition 9 (Directed Acyclic Graphs). *G is a Directed Acyclic Graph (DAG) if and only if G is a directed graph without directed cycles.*

Definition 10 (Reachability). *Vertex $v \in V$ is reachable from $u \in V$ in graph G if there is a directed path from u to v.*

Definition 11 (Transitive Reduction of a Directed Acyclic Graph). *The transitive reduction of a directed acyclic graph $G = (V, E)$ is a graph $G^- = (V, E^-)$ with the fewest possible edges where nodes in G and G^- have the same reachability.*

Definition 12 (Transitive Closure of a Directed Graph). *The transitive closure of a directed graph $G = (V, E)$ is the graph $G^+ = (V, E^+)$ where $E^+ = \{(u, v) : u \in V, v \in V \text{ and } v \text{ is reachable from } u \text{ in } G\}$.*

In previous work, the concept of transitive closure is associated to the relation R on V, pair of vertices in the edge set E, i.e. $(u, v) \in E \Leftrightarrow uRv$. The transitive closure of a homogeneous binary relation R on set V is defined as the smallest (with respect to the set inclusion $\subseteq$) transitive relation on V where $R \subseteq R^+$. Then, it would be easy to show that the set of oriented pairs of all reachable vertices, namely $\{(u, v) : u \in V, v \in V \text{ and } v \text{ is reachable from } u \text{ in } G\}$, is the transitive closure of the relation induced by the edges of a graph. However, we just define the transitive closure of a graph as is. Additionally, if R is transitive, $R = R^+$ by definition.

Definition 13 (Direct and Transitive edges in a DAG). *Let $G = (V, E)$ be a DAG, and let $G^+ = (V, E^+)$ and $G^- = (V, E^-)$ be its transitive closure and transitive reduction, respectively. We call edges in E^- direct or non-transitive edges, and edges in $E^+ \backslash E^-$ transitive edges.*

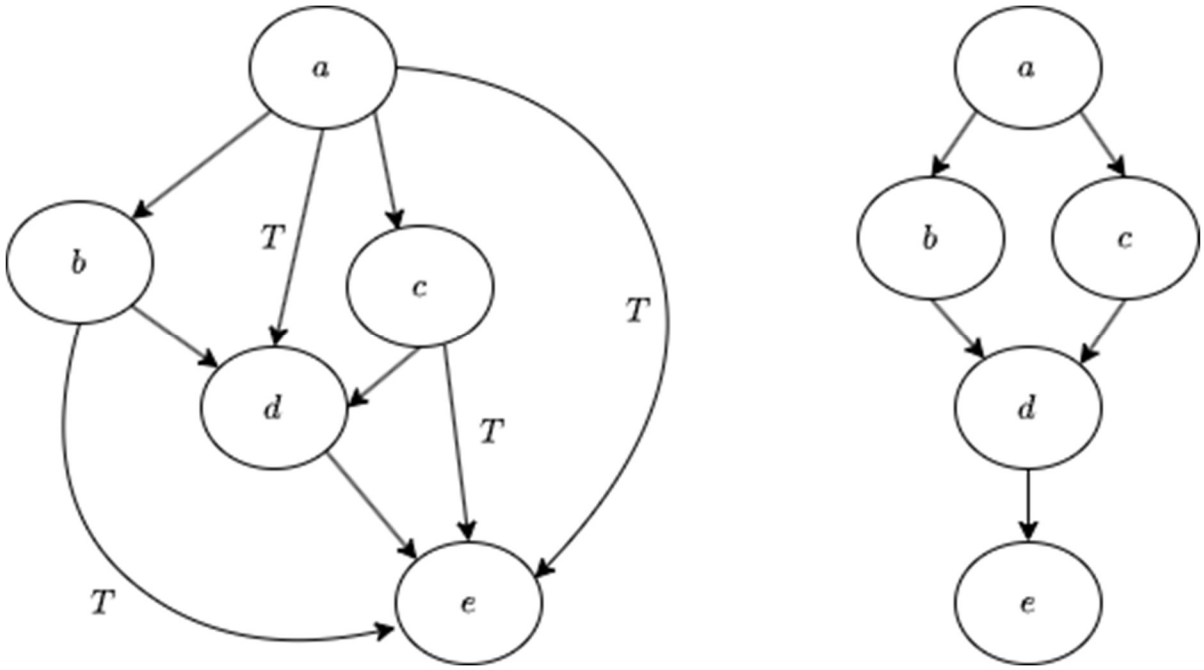


Figure 2.1: Graph representations of the transitive closure (left) and the transitive reduction (right) of the same DAG. Transitive edges are marked with a T.

Figure 2.1 is an example of the transitive closure and the transitive reduction of the same graph. It is easy to show that the transitive reduction of the graph in Figure 2.1 is minimal and unique. More generally, the transitive reduction of a finite directed acyclic graph G is unique, implying that **in a DAG** there is only one way of splitting edges between direct edges and transitive edges. Fortunately, because prerequisite relations define a partial order on concepts (**Proposition 2**), its graph representation is a DAG (**Proposition 3**). Therefore, in the context of our work, distinguishing between transitive edges and direct edges is possible because edges can't be both. We base our Graph Split Algorithm on this property (refer to Section 3.2.2).

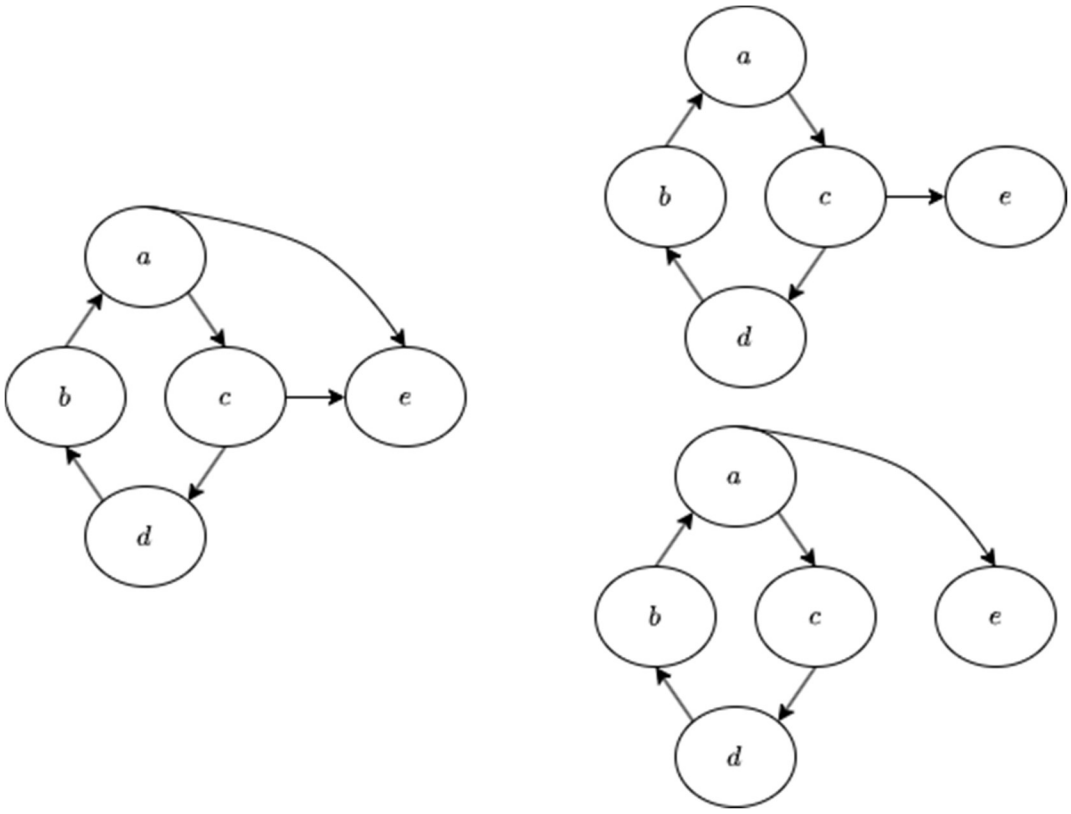


Figure 2.2: Graph representations of two possible transitive reductions (right) of a directed graph with a cycle (left).

However, it is important to note that, as shown in Figure 2.2, the uniqueness property of the transitive reduction does not hold when the directed graph contains a cycle. In fact, because of the cycle between edges (a, b, c, d), removing edges (c, e) or (a, e) does not affect the reachability of the graph. This highlights the importance of verifying that prerequisite relation datasets satisfy the required properties of a strict partial order, which is done in Section 3.1.4.

Definition 14 (Weakly connected components in a directed graph). *Let $G = (V, E)$ be a directed graph. A weakly connected component $WCC = (V', E')$ is a subgraph of G where for any given pair of nodes $(u, v) \in V' \times V'$ there is a path between u and v in the underlying undirected graph.*

Definition 15 (Strongly connected components in a directed graph). *Let $G = (V, E)$ be a directed graph. A weakly connected component $SCC = (V', E')$ is a subgraph of G where for any given pair of nodes $(u, v) \in V' \times V'$ u is reachable from v and v is reachable from u.*

Remarks:

- It is important to note that there are no strongly connected components in a DAG. In fact, DAGs do not contain cycles, hence there cannot be two edges u, v where we can go from u to v and then come back to u, as that would be a cycle.
- Consequently, in DAGs, only weakly connected components are relevant to count as they are the only indicator of graph connectivity (see Section 3.2.5 for statistics on AL-CPL).

Proposition 3. *Let R be a strict partial order on set S. Let $G = (V, E)$ be the directed graph associated to the strict partial order R on V where $V = S$ and $E = \{(x, y): x \in V, y \in V \text{ and } xRy\}$. Then G is a directed acyclic graph without self-loops where $G = G^+$.*

Proof:

Let $G = (V, E)$ be the graph associated to the strict partial order R on S where $V = S$ and $E = \{(x, y): x \in S, y \in S \text{ and } xRy\}$.

- G is a directed graph:

By definition of G, the edge set E consists of ordered pairs from $V \times V$, i.e. $E \subseteq V \times V$. According to **Definition 5**, G is a directed graph.

- G is acyclic:

To prove that the graph is acyclic, we will exploit the irreflexive and transitive nature of the strict partial order and proceed by contradiction.

Let's assume, for the sake of contradiction, that there exists a cycle in G starting from $v_1 \in V$. Meaning that there exists a sequence of vertices $(v_1, \dots, v_n)$ in V such that $(v_1 R v_2, \dots, v_n R v_1)$. By transitivity property of R, we get that $v_1 R v_1$ which is a contradiction because R is irreflexive. Therefore, our assumption of the existence of a cycle is false, and G is acyclic.

- No self-loops:

The irreflexive property of R ensures that no element in V is related to itself. This means there are no edges of the form $(x, x) \in E$. Therefore, G has no self-loops.

- Let's prove that $\boldsymbol{G = G^+}$:

By definition $E \subseteq E^+$, we only need to prove $E^+ \subseteq E$.

Let $e = (u, v) \in E^+$ be an edge of the transitive closure of G called $G^+ = (V, E^+)$. By the definition of E^+, v is reachable from u in G. Therefore, there exists a sequence of vertices $(u, v_1, \dots, v_n, v)$ in G such that there exists a directed path from u to v $((u, v_1), \dots, (v_n, v))$. By the definition of E, edges exist when there is a relation between the vertices: $(uRv_1, \dots, v_nRv)$. By the transitivity property of R, we get that uRv, thus $e \in E \Rightarrow E = E^+$.

Q.E.D.

Proposition 4. *Let G be a directed acyclic graph without self-loops and G^+ its transitive closure. Let R be the relation on V where $xRy \Leftrightarrow (x, y) \in E^+$. Then R is a strict partial order on V.*

Proof:

Because the proof of **Proposition 4** is sensitively similar to the proof of **Proposition 3**, only the idea of the proof will be explained:

- First, the acyclic property of the graph implies the asymmetry of the relation.
- Second, the absence of self-loops implies that the relation is irreflexive.
- Finally, because $\boldsymbol{xRy \Leftrightarrow (x, y) \in E}$ and $\boldsymbol{G}$ is transitive, $\boldsymbol{R}$ is transitive. Therefore, $\boldsymbol{R}$ is a strict partial order on V.

Q.E.D.

Propositions 3 and **4** imply that we can go back and forth between a strict partial order and its directed acyclic graph representation without losing their mathematical properties. In the following sections, we will use graph representations of prerequisite relations between concepts to illustrate examples as it is more practical.

Definition 16 (Union of binary relations). *Let R and R' be two binary relations on set S and S'. We call $R \cup R'$ the union of both binary relations R and R' where $R \cup R' = \{(x, y): xRy \text{ or } xR'y\}$.*

Proposition 5. *The union of two strict partial orders is not a strict partial order.*

Proof:

To prove this proposition, we only need one counter example. But for the sake of clarity, we are going to build two counterexamples, one that transgresses transitivity, and the other one that transgresses asymmetry.

- First counterexample:

Let $S = \{a, b, c\}$ be the set of three elements, and R and R' be two strict partial orders on S where $R = \{(a, b)\}$ and $R' = \{(b, c)\}$.

Then, $R \cup R' = \{(a, b), (b, c)\}$ and $R \cup R'$ is not transitive because it is missing (a, c).

- Second counterexample:

Let $S = \{a, b, c\}$ be the set of three elements, and R and R' be two strict partial orders on S where $R = \{(a, b)\}$ and $R' = \{(b, a)\}$.

Then, $R \cup R' = \{(a, b), (b, a)\}$ and $R \cup R'$ is not asymmetric.

Q.E.D.

Proposition 6. *The union of two strict partial orders on disjoint sets is a strict partial order.*

Proof:

The proof is straight forward, using **Proposition 1** we only need to prove that the union is irreflexive and transitive.

Let R_1, R_2 be two strict partial order relations on disjoint sets S and S' where $S \cap S' = \emptyset$. Let R be the union of R_1, R_2. Let us prove that R is a partial order on $S \cup S'$.

- Irreflexivity:

If $x \in S \cup S' \Rightarrow x \in S \ \wedge x \notin S' \vee \ x \notin S \ \wedge x \in S'$ because S and S' are disjoints.

Case 1: If $x \in S \wedge x \notin S'$ then $\neg(xR_1x)$ because R_1 is a strict partial order and $\neg(xR_2x)$ because $x \notin S' \Rightarrow \neg(xRx)$.

Case 2: If $x \notin S \ \wedge x \in S'$, then $\neg(xRx)$ (same reasoning).

- Transitivity:

Let $x, y, z \in S \cup S'$ and $xRy \wedge yRz$. This proof by exhaustion has 4 cases.

Case 1 and 2: both relations are the same, i.e. $xR_1y \wedge yR_1z$ or $xR_2y \wedge yR_2z$. Because both R_1 and R_2 are transitive, we get xRz.

Case 3 and 4: relations are different, i.e. $xR_1y \wedge yR_2z$ or $xR_2y \wedge yR_1z$. It requires $y \in S \cap S' = \emptyset$, which is impossible.

Only case 1 and 2 are possible, and in both cases, we get xRz. Therefore, if $xRy \wedge yRz \Rightarrow xRz$.

Moreover, note that since S and S' are disjoint, we avoid both counterexamples given in the proof of **Proposition 5**.

Q.E.D.

Propositions 5 and **6** are important to keep in mind when merging two concept prerequisite relations. In order for the merged relations to represent a strict partial order, one must compute their transitive closure and add the missing directed edges to the merged graph. One must also ensure that for the intersecting concepts, there are no relations that violate the asymmetry property in the union (i.e. A→B and B→A). If so, domain experts must break the tie: they must decide whether concept A is a prerequisite for concept B or vice versa.

In this work, we follow these guidelines when we merge two graphs associated with concept prerequisite relations in order to obtain a graph associated with a strict partial order as a result.

In summary, our work is based on the theoretical background of concept prerequisite relations viewed as a strict partial order. This allows us to represent the prerequisite relations between concepts using DAGs (see **Proposition 3**), which is done in Section 3.1.2. This graph representation allows us to train our models (see Section 4.1), and highlights a methodological problem related to how previous work has divided their data for their CPL experiments (see Section 3.2.1). Finally, in Section 3.2.2, we introduce a solution to this methodological problem. In the following section, we will introduce the background related to semantic representations in natural language processing.

2.2 Neural Network Architectures for Word and Sentence Embeddings

This section provides the background and context on embeddings. An embedding of a word is a learned vector representation of its meaning based on its distribution in textual data (Jurafsky & Martin, 2024). A sentence embedding is similar to a word embedding but for an entire sentence. Word embedding methods learn a static single vector embedding for each unique word (Bojanowski et al., 2017; Mikolov et al., 2013). Whereas contextual embeddings capture semantic relationships between words based on their position in the text (Devlin et al., 2019; Reimers & Gurevych, 2019). This type of embedding is not static and changes depending on the context in which the word is used.

This section is divided into two subsections. First, we introduce the fully connected layer, which is the foundational block of word embeddings, and present how the word embeddings are trained in Section 2.2.1. Then, we introduce the Bidirectional Encoder Representations from Transformers called BERT (Devlin et al., 2019), which is the neural network architecture behind the contextual embeddings, and explain how it can be used to generate contextual word embeddings and contextual sentence embeddings in Section 2.2.2.

2.2.1 Word2Vec and FastText Word Embeddings

A **fully connected layer** is the elementary block of the feedforward network used to learn word embeddings (see Figure 2.3). It takes as input a vector $X \in \mathbb{R}^n$ and returns the output $y_{out} = f(AX + b)$, where $A \in \mathbb{R}^{d \times n}$ and $b \in \mathbb{R}^d$ are the weights and the biases of the fully connected layer, $d \in \mathbb{N}$ is the size of the output, and f is the differentiable activation function. Figure 2.3 illustrates what a fully connected layer is. It is called a "fully connected" layer or sometimes a "densely connected" layer, because each input of the input vector influences each output of the output vector if its associated weight is not zero.

A **feedforward neural network** is a neural network consisting of one or more fully connected layers. It forms the backbone of both the Word2Vec (Mikolov et al., 2013) and FastText (Bojanowski et al., 2017) neural network models for generating word embeddings.

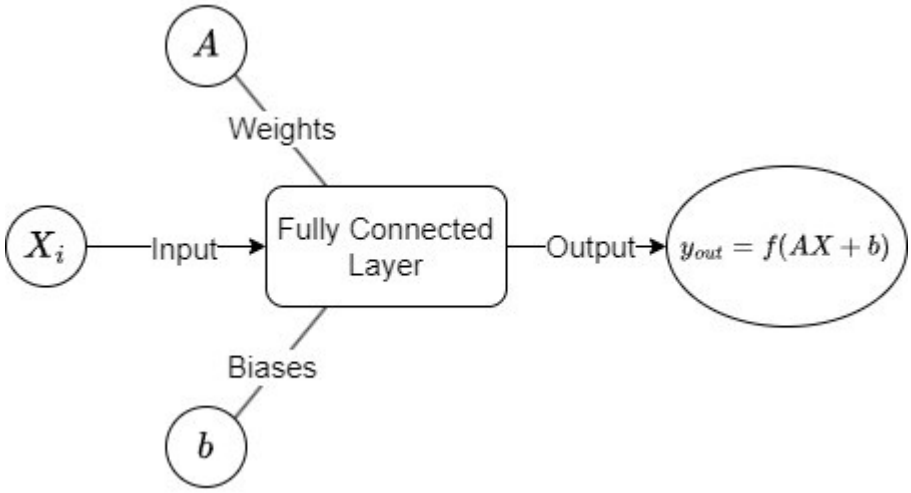


Figure 2.3: Representation of a fully connected layer with its weights, biases, inputs and outputs.

Mikolov et al. (2013) introduced Word2Vec, which has two variations: Continuous Bag of Words (CBOW) and Skip-Gram. Both models use a single fully connected layer. The CBOW model predicts the current word based on its surrounding words, while the Skip-Gram model predicts the context given the current word (refer to Figure 2.4).

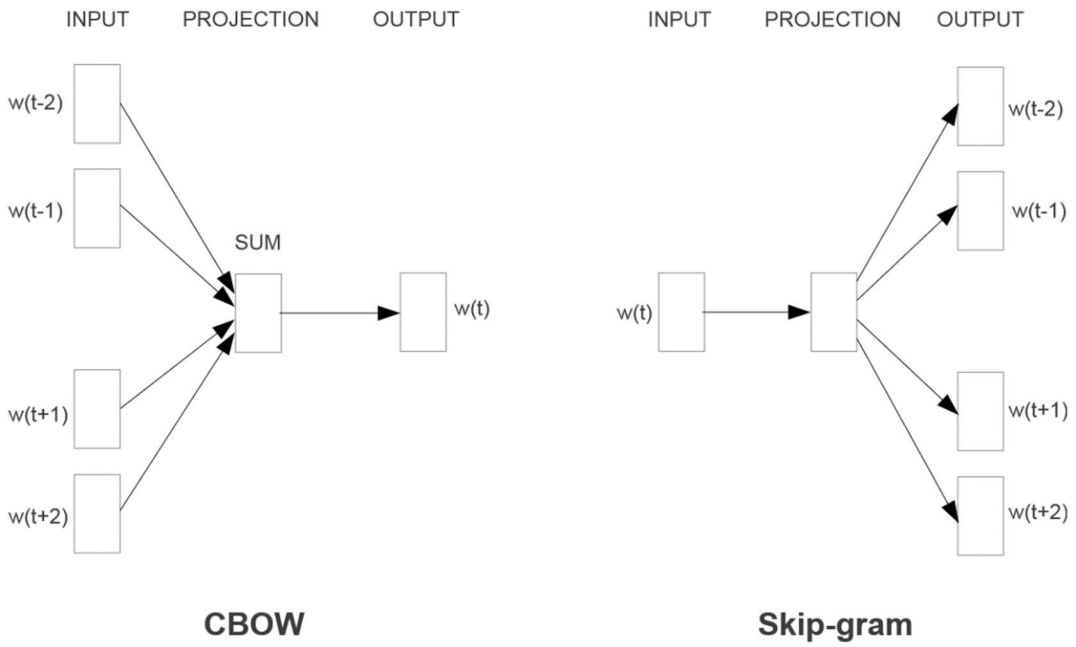


Figure 2.4: CBOW and Skip-gram model architectures. CBOW uses the context to predicts the current word, and the Skip-gram uses the current word to predict the context. Figure from Mikolov et al. (2013).

One limitation of Word2Vec is its handling of words absent from its training set. It assigns the same embedding to all unknown words, resulting in a vector representation that does not capture the meaning of the unknown word based on its distribution in textual data.

To address this issue, FastText expands on the Word2Vec model by training the Skip-gram model not only on words but also on subwords or character n-grams (Bojanowski et al., 2017). A character n-gram is a sequence of n characters. For instance, with n = 3 the word ‘Machine’ with the special boundary symbols ‘<’ and ‘>’, ‘<Machine>’, would be represented by the following sequence of character n-grams:

‘<Ma, Mac, ach, chi, hin, ine, ne>’.

Then a Skip-gram embedding is learned for each constituent n-gram, and the word 'Machine' is represented by the sum of all of the embeddings of its constituent n-grams. This approach allows FastText to capture the internal structure of words, making it proficient in handling morphologically rich languages, rare words, and even misspelled words.

However, using this method to represent concepts has its limitations. FastText computes the average word embeddings of each word in a short sentence to generate a sentence embedding. Nevertheless, concepts like 'Machine Learning' are comprised of two words, ‘Machine’ and ‘Learning’, which have different meanings when used separately. Therefore, when used together in a sentence, their combined meaning may not be accurately represented by the average of their individual word embeddings. Indeed, according to Wikipedia in English, which is used to pretrain FastText in English (Bojanowski et al., 2017), a machine is 'a physical system using power to apply forces and control movement to perform an action' and learning is 'the process of acquiring new understanding, knowledge, behaviors, skills, values, attitudes, and preferences'. This highlights the need for a more effective generate embeddings for concepts. This motivates our exploration of contextual embeddings as they capture semantic relationships between words based on their position in the text.

2.2.2 BERT-based Contextual Embeddings

This section introduces BERT (Devlin et al., 2019), an encoder-based transformer used for various natural language processing tasks. The encoder consists of multiple layers, each containing two sub-layers: a multi-head self-attention mechanism and a fully connected feed-

forward network. The self-attention mechanism enables the model to weigh the importance of different words within a sentence, allowing it to capture the context more effectively. Indeed, the self-attention mechanism of the encoder grants BERT the ability to read the entire sequence of words at once instead of reading it from left to right, making it bidirectional. This allows BERT to learn the context of a word by looking at the words surrounding it. BERT's ability to build efficient embeddings from text is due to its pre-training on large corpora. BERT is pre-trained on two tasks: masked language modeling and next sentence prediction.

Masked Language Modeling. BERT is pre-trained on the masked language modeling task. This task involves randomly masking some of the tokens from the input. The objective is for the model to predict the original identity of the masked words based solely on their context. This training task enables BERT to learn a deep understanding of language structure and context. During the training phase, the model predicts the masked word based on the context provided by the other non-masked words in the sequence. This process involves replacing approximately 15% of the words in each sentence with a '[MASK]' token, and predicting the masked word using the provided context (Devlin et al., 2019). The loss function takes only the predictions on masked words into account.

Next Sentence Prediction. BERT is also pretrained on a next sentence prediction task where the model is presented with pairs of sentences and is trained to predict whether the second sentence in the pair is the actual subsequent sentence in the original document, or if it's a random sentence from the corpus. During training, half of the input pairs consist of a first sentence followed by its true next sentence, while the other half consists of a first sentence followed by a random sentence from the corpus that is not related to the first. To differentiate between the two sequences, special tokens are added to the sentence tokens: [CLS] at the beginning of the tokens, [SEP] between the two sequences, and [SEP] at the very end. This task requires BERT to understand the coherence and flow of information in texts, allowing it to capture relationships across sentences. Both masked language modeling and next sentence prediction tasks are trained altogether. The resulting training loss function is the sum of the loss functions of both tasks.

Two popular methods for generating sentence embeddings are to average the BERT output layer (known as BERT embeddings) or to use the output of the '[CLS]' token. However, Reimers &

Gurevych (2019) showed that these methods produce sentence embeddings are outperformed by the average of word embeddings. Thus, we wish to avoid such methods.

Therefore, in the next paragraph, we introduce the method proposed by Reimers & Gurevych (2019) for generating sentence embeddings using Sentence-BERT.

Sentence-BERT

This section introduces Sentence-BERT, a method developed by Reimers & Gurevych (2019) to train two BERT networks with tied weights (siamese network structure) to accurately measure the semantic similarity between two sentences. This method is designed to enhance sentence-level embedding generation.

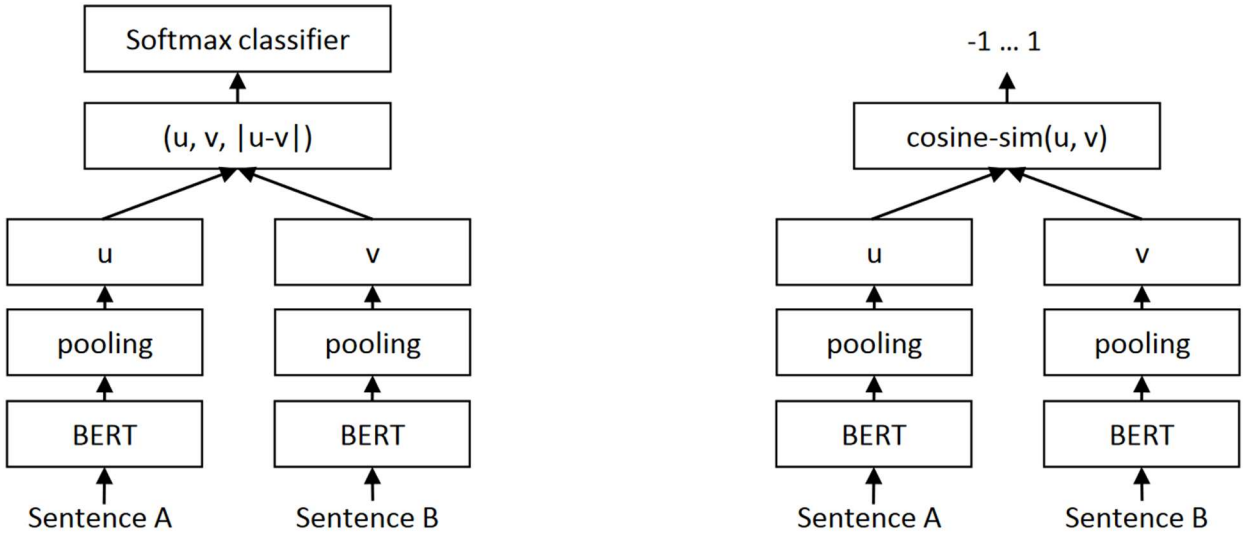


Figure 2.5: Sentence-BERT architecture with classification objective function (left) and for the the regression objective function (right). Figure from Reimers & Gurevych (2019).

To achieve this, Reimers & Gurevych (2019) trained Sentence-BERT on datasets that include pairs of sentences with annotations regarding their semantic similarity or relationship.

They used the Stanford Natural Language Inference (SNLI) corpus, a collection of 570,000 human-written English sentence pairs, and the Multi-Genre Natural Language Inference (MultiNLI) corpus, a crowd-sourced collection of 433,000 English sentence pairs. Both datasets have annotated sentence pairs with three possible labels: entailment, contradiction, or neutral.

Reimers & Gurevych (2019) experiment with two main objective functions:

Classification Objective Function (Figure 2.5 on the left**).** Sentence embeddings u and v are concatenated with their difference $|u - v|$ and are multiplied with a trainable weight. The cross-entropy loss is minimized for classification.

Regression Objective Function (Figure 2.5 on the right**).** The cosine similarity between the two sentence embeddings u and v is computed and the mean squared error is used as the loss function.

The classification objective function is used for NLI and yields great results.

The main distinction between BERT and Sentence-BERT is that Sentence-BERT can generate semantically meaningful and computationally efficient sentence embeddings for direct comparison, whereas BERT relies on a computationally intensive pairwise comparison method for sentence-level tasks. This feature makes Sentence-BERT ideal for applications that require fast and precise semantic similarity evaluations between sentences, which is precisely what we need for CPL.

2.3 Graph Convolutional Networks Explained

Recently, GCNs have gained in popularity in the field of CPL. The solution for CPL presented in this thesis is based on GCNs. This section explains what is a GCN based on the original paper and version of GCN presented by Kipf & Welling (2017).

GCNs are designed to learn node representations on a graph $G = (V, E)$. They can generate node representations in a semi-supervised manner that are useful for tasks unrelated to their training task (Kipf & Welling, 2017). They update node representations by aggregating information from neighboring nodes. The GCN takes two inputs:

- The initial feature matrix $X \in \mathbb{R}^{N \times D}$ of dimensions $N \times D$, where $N = |V|$ is the number of nodes in G and D is the dimension feature vectors for each node. Each $X_i \in \mathbb{R}^D$ is the feature vector of dimension D representing the node $i \in V$ in graph G.
- The graph is usually represented as an adjacency matrix $A \in \mathbb{R}^{N \times N}$, where $A_{ij} = 1$ if and only if $(i, j) \in E$.

Moreover, the adjacency matrix A is invariant throughout the layers. However, the hidden feature matrix at layer l, H^l, representing node embeddings, changes and is updated by each layer following this simple update rule:

$$H^{l+1} = f(H^l, A) = \sigma\left(\widehat{D}^{-\frac{1}{2}}\hat{A}\widehat{D}^{-\frac{1}{2}}H^l W^l\right), \tag{2.1}$$

where $\hat{A} = A + I$ is the modified adjacency matrix to avoid numerical instabilities during training (Kipf & Welling, 2017), and $\widehat{D}$ is the diagonal node degree matrix of $\hat{A}$ ($\widehat{D}_{ii} = \sum_j \hat{A}_{ij}$), H^{l+1} is the output of the l-th GCN layer ($H^0 = X$), W^l is the weight matrix for the l-th GCN layer and $\sigma(.)$ is a non-linear activation function.

When stacking multiple GCN layers, the new node embeddings at layer l include embeddings of nodes up to a maximum distance equal to l in their embeddings. Indeed, a simple way to visualize this is through inductive reasoning: at layer one the updated node representations are the result of the product of the adjacency matrix, the initial node representations and the weight matrix. Therefore, at layer one, only neighbors of the node are included in its new representation. However, at layer two, the updated node representations are the result of the product of the adjacency matrix, the layer one node representations which include neighbors of each node, and

the weight matrix. Consequently, node embeddings at layer two include the neighbors of neighbors, which are nodes at distance 2, in their new embeddings. Thus, at layer l, node embeddings include nodes up to a maximum distance equal to l.

Equation 2.1 is the result of a first-order approximation of spectral graph convolutions (Kipf & Welling, 2017), hence the Convolution in Graph Convolutional Network.

In summary, at each layer, the GCN updates its representations for each node in the graph by aggregating information from neighboring nodes. These node representations can be used for node classification or for transductive link prediction (see definitions below). In this thesis, GCNs are both used for node classification and transductive link prediction. The following section defines the link prediction task and the transductive link prediction setup.

2.3.1 Transductive Link Prediction

In this section, the definitions and facts are inspired and taken from the work of Ali et al. (2021).

Definition of Link Prediction. Let $G = (V, E)$ be a graph where V is the set of vertices and $E \subseteq V \times V$ is the set of edges, also called "true links", across the nodes of the graph G. Let $G' = (V', E')$ be a subgraph of G where $V' \subseteq V$ and $E' \subset E$ is a subset of true links called "observed links" or "message passing edges". The objective of link prediction is to predict the unobserved true links, albeit the remaining true links that are not present in E'.

Link Prediction as a binary classification task. In the binary classification formulation of link prediction, potential links are classified as either true links (1) or false links (0). In this setting, the link prediction model G_l learns to classify links from a chosen subset $E' \subset V \times V$ of edges into positive and negative labels: $G_l: E' \mapsto \{0,1\}$.

Additionally, there are two main scenarios for link prediction, they are called transductive link prediction and inductive link prediction.

In the **transductive setup**, $V = V'$ and we perform link prediction over the same nodes of the graph seen at training time, but on different edges. However, transductive link prediction does not allow models to process unseen nodes.

Recent work in the field of link prediction has focused on a less restrictive setup in which models can process unseen entities. This setup is called the inductive setup. In the **inductive setup**,

inference can be made on a new graph totally disconnected from the training graph, or on a larger graph that includes the training graph.

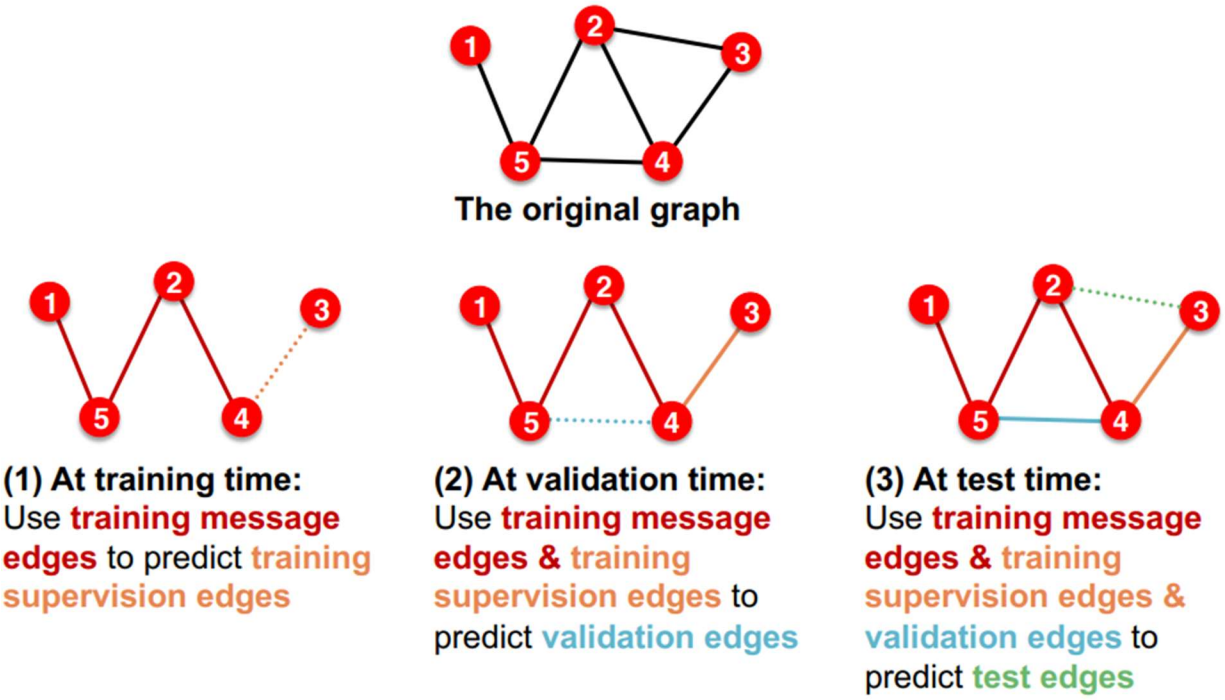


Figure 2.6: An example of a transductive link prediction split on a graph. Figure from Jure Leskovec's slides[2].

In this thesis, we consider CPL to be a link prediction task between concepts, and we solve it in the transductive setup as a binary classification task. Therefore, in the remaining of this section we are going to explain precisely how the data is split for transductive link prediction. Indeed, there is a specific way of splitting the data recommended by Leskovec for transductive link prediction with GCNs illustrated in Figure 2.6, and detailed in the paragraph below:

- **Training Message Edge:** This split includes the edges present in the training graph used as input for the GCN model. These edges are used only for message passing during training, meaning the model does not use these edges as ground truth to update its weights and parameters.
- **Training Supervision Edge:** This split involves holding out edges in the training set that are used as ground truth to train the model and update its weights. By excluding these

[2]His slides are available at https://snap.stanford.edu/class/cs224w-2020/slides/08-GNN-application.pdf (see slide 68).

edges, the model is forced to generalize and learn from the remaining graph structure which it has not seen in its input graph. Additionally, we use training message edges and training supervision edges as input to predict validation edges.

- **Validation Edge:** This split is dedicated to evaluating the performance of the model. It includes edges that are held out from both the training message and supervision edges. They are used to assess the model's generalization and its ability to make accurate predictions on unseen data. Model performance on validation is used to choose the best model and the hyperparameters that go along with it. We use training message edges, training supervision edges, and validation edges to predict test edges.
- **Test Edge:** It contains edges that have been held out from all previous splits. They assess the model's effectiveness in real-world scenarios.

A practical example of how we split the data for CPL is illustrated and explained in detail in Section 4.2.1.

2.3.2 R-GCN

Relational GCNs are modified GCNs to work with graphs with different types of relationships and links between nodes (Schlichtkrull et al., 2017). To illustrate, let us consider a graph representing relations between customers, stores, and suppliers. In this scenario, models should be able to distinguish between the edges linking customers and stores, and the edges linking stores and suppliers as they do not hold the same semantic meaning.

Schlichtkrull et al. (2017) change the update equation from Equation 2.1 to include distinct weight matrices for each type of relation:

$$H^{l+1} = \sigma\left(\sum_{r=1}^{m} {D_r}^{-\frac{1}{2}} A_r {D_r}^{-\frac{1}{2}} H^l W_r^l + W_0^l H^l\right), \tag{2.2}$$

where A_r is the adjacency matrix related to the relation r, D_r is the diagonal node degree matrix of A_r, H^l is the hidden representation at layer l, W_r^l is the weight matrix for relation r for the l-th R-GCN layer, and $W_0^l H^l$ denotes the self-connection of nodes and is treated as a special relation type for each node in the data (Schlichtkrull et al., 2017). This self-connection enables a node at layer $l + 1$ to be informed by the corresponding representation at layer l. It is similar to Kipf & Welling's (2017) approach regarding the modified adjacency matrix $\hat{A} = A + I$. The result is the

same, which means that each node at layer $l + 1$ can use its own representation at layer l to compute its updated representation.

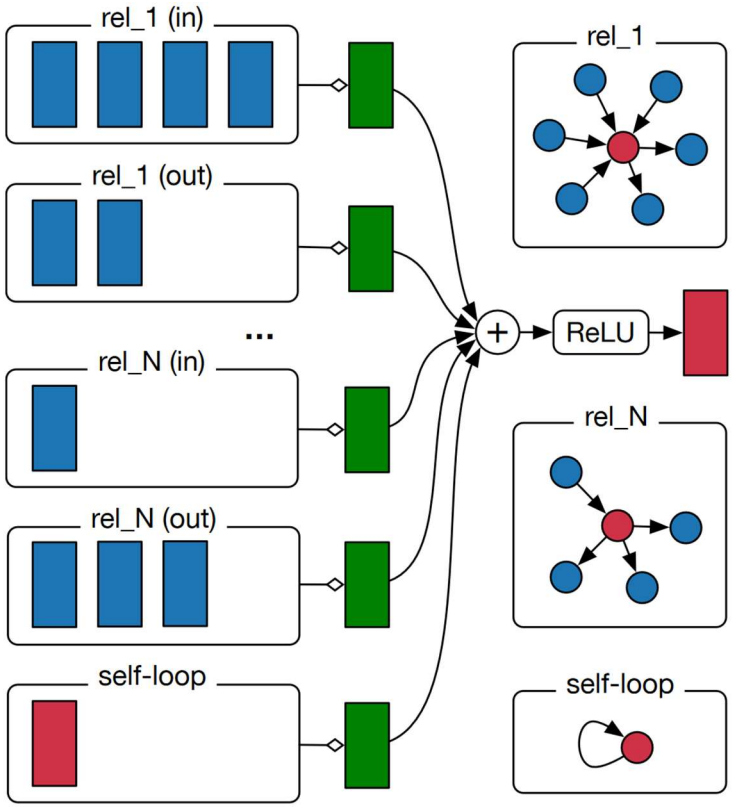


Figure 2.7: Illustration of the update of a single node representation (red) in an R-GCN layer. Figure from Schlichtkrull et al. (2017).

In summary, R-GCN aggregates nodes from different relational graphs using distinct weight matrices (in green) as illustrated in Figure 2.7. Approximately, one can conceptualize R-GCN as a collection of multiple GCNs applied to different relational graphs, with their outputs subsequently aggregated.

In this thesis, R-GCNs are used to distinguish between two types of links: prerequisite and non-prerequisite relations between concepts. More on that in Section 4.1.3.

2.4 Concept Prerequisite Learning

Numerous research used text to solve CPL as a binary classification task between pairs of concepts (Bai et al., 2021; Chaplot et al., 2016; Liang, Ye, Wang, et al., 2018; Miaschi et al., 2019; Pan et al., 2017; Roy et al., 2019; Zhou et al., 2020). Researchers used publicly available

datasets that sourced their textual data from online resources like MOOCs (Chaplot et al., 2016; Pan et al., 2017; Roy et al., 2019) or Wikipedia (Bai et al., 2021; Liang, Ye, Wang, et al., 2018; Miaschi et al., 2019). Early studies based on MOOCs used statistics and semantic relatedness to infer prerequisites between concepts (Chaplot et al., 2016; Pan et al., 2017). Others, like Roy et al. (2019), employed a pairwise-link LDA model and a siamese network to discover unknown prerequisite relations in MOOCs.

Additionally, Miaschi et al. (2019) define two distinct settings for CPL: the in-domain setting and the cross-domain setting. To solve CPL in the in-domain setting, models must be trained and tested on concept pairs belonging to the same domain. And the evaluation must be performed using a k-fold cross-validation. In contrast, the cross-domain setting requires models to be trained on all domains except one that has been left out for the test set.

2.4.1 CPL as a Machine Learning Task

This thesis focuses on supervised CPL using the AL-CPL dataset, which was derived from Wikipedia by Wang et al. (2016) and further enhanced by Liang, Ye, Wang, et al. (2018)[3]. Previous research used textual features and graph-based features from Wikipedia to train machine learning models to identify these relations (Liang, Ye, Wang, et al., 2018). Later, Zhou et al. (2020) improved upon this using an AdaBoost approach. The current best-performing method on AL-CPL combines word embeddings and sixteen features mined from text such as the asymmetric RefD metric (Liang et al., 2015), which measures the asymmetry of the references between two concepts in their respective Wikipedia pages (Miaschi et al., 2019). They use Word2Vec (Mikolov et al., 2013) to represent the first 400 words of the Wikipedia page associated with each concept, then feed these embeddings to an LSTM, followed by a dense layer for CPL. We refer to it as the state-of-the-art approach (SOTA). Bai et al. (2021) used a similar approach to Miaschi et al. (2019) with BERT (Devlin et al., 2019) instead of Word2Vec, but they couldn't surpass Miaschi et al.'s (2019) results on AL-CPL.

Overall, past works relying on textual data did not make use of the information provided by the graph structure associated with the prerequisite relations between concepts.

[3] Available at https://github.com/harrylclc/AL-CPL-dataset.

2.4.2 CPL as a Link Prediction Task

In contrast, newer work addressed CPL as a link prediction task by building a graph based on existing prerequisite relations. Graph-based approaches are trained on publicly available datasets sourced from MOOCs. Li et al. (2018) developed the first concept graph approach on the LectureBank dataset and used a VGAE to solve CPL as a link prediction task. Their study highlights the limits of only relying on concept prerequisites for inference. To illustrate, Graph Neural Networks (GNNs) rely on a node's incoming and outgoing edges to its neighbors to form its representation. Therefore, using a GNN for a concept without any prerequisite relations, i.e. without incoming nor outgoing edges, defeats its purpose. This is an important limitation to consider when 15% of AL-CPL's concepts have no prerequisites. Following Li et al. (2018), subsequent methods tackled this challenge by developing a heterogeneous graph for CPL called the resource-concept graph, as seen in works such as (Li et al., 2020; Tang et al., 2023; Zhang et al., 2022). The resource-concept graph includes multiple relationships between two types of nodes: resource, such as a course, and concept. Tang et al. (2023) builds its own resource-concept graph by mining course syllabi available in MOOCs. If a concept is mentioned in the syllabus, then the course and the concept will be linked whereas links between courses are prerequisites. However, this approach will not hold for CPL datasets without prerequisites from multiple resources, or resources that cannot be linked. This motivates our choice of restraining our work to concept graphs as they are easier to mine and more applicable to all datasets.

Moreover, Li et al. (2020), Zhang et al. (2022) and Tang et al. (2023) combined graph structures with textual data for CPL. For instance, Li et al. (2020) used Phrase2Vec (Wu et al., 2020) to generate concept embeddings based on their names, while Tang et al. (2023) used their continually pre-trained BERT model for the same purpose. Interestingly, Li et al. (2020) found that their BERT embeddings, derived by averaging token representations of the concept's label, didn't surpass the performance of Phrase2Vec embeddings. On the other hand, Tang et al. (2023) used the `[CLS]` token to generate embeddings for concept pairs using inputs like `[CLS] Conditional probability [SEP] Markov chain [SEP]` , without adding any additional context to the concepts. Tang et al. (2023) reported promising results on multiple MOOC CPL datasets. Unfortunately, we did not have access to their model or code to reproduce their results.

2.4.3 Limitations of Previous Work

Previous work has misused transformers in the context of CPL. They have either been used with ML architectures that do not synergize well with transformers, or they have created concept representations in a sub-optimal way, according to natural language processing works such as Reimers & Gurevych (2019).

Indeed, BERT, as a bidirectional Transformer, emphasizes the context of a token over its position in the sequence. In contrast, LSTMs excel with sequential data, such as time series (Hua et al., 2019), making them more suited for word2vec embeddings where the word order is crucial compared to BERT embeddings. Thus, we claim that Bai et al. (2021) use inefficiently BERT embeddings, which explains why their results are inferior to Miaschi et al. (2019).

Additionally, we suspect that previous work using Tranformers did not leverage the context of concepts to generate effective concept embeddings. Indeed, the way BERT embeddings have been computed in Li et al. (2020) and Tang et al. (2023) is sub-optimal: Li et al. (2020) computed the element-wise average of BERT embeddings for each token of the concept's name and Tang et al. (2023) fed as input to BERT sentences structured as '[CLS] concept_name_1 [SEP] concept_name_2 [SEP]' where the `[CLS]` token embedding was used to represent the concept pair. However, Reimers & Gurevych (2019) showed that averaging BERT's output layer, known as BERT embeddings, or using the output of the `[CLS]` token yields rather bad sentence embeddings, often worse than averaging GloVe embeddings (Pennington et al., 2014). In addition, simply using concept names outside of the context of a meaningful sentence, such as "circle [CLS] plane", fails to benefit from the contextualized embeddings generated by transformers.

Furthermore, CPL datasets have a higher number of non-prerequisite relations than prerequisites (Li et al., 2018; Liang et al., 2017; Liang, Ye, Wang, et al., 2018; Roy et al., 2019). None of the graph-based methods mentioned earlier include non-prerequisite relations in their graph structure. Non-prerequisite relations are identified when concept A is not a prerequisite for concept B. In the AL-CPL dataset, 15% of concepts are linked only by non-prerequisite relations, leading to their exclusion from graph representations in previous methods. In our approach, we include non-prerequisite relations in our graph structure, which make up 69.34% of the relations in the AL-CPL dataset.

Finally, to date and to our knowledge, all previous CPL approaches have relied on the k-fold cross-validation procedure to split their datasets into a training and a test split (Bai et al., 2021; Li et al., 2018, 2020; Liang, Ye, Wang, et al., 2018; Miaschi et al., 2019; Pan et al., 2017; Tang et al., 2023; Zhang et al., 2022; Zhou et al., 2020). K-fold cross-validation involves randomly splitting the dataset into k folds of approximately equal size, where each unique fold will be selected as the test set, and the remaining $k - 1$ folds as the training set, thereby generating k different training and test sets (James et al., 2013, Page 181). The act of randomly splitting the dataset into k folds is called the random split. This random split assumes that relations in the dataset are independent and does not account for the transitivity property of prerequisite relations (refer to **Definition 3** in Section 2.1). For instance, if the training set has the relations $A \rightarrow B$ and $B \rightarrow C$ and the test set has the relation $A \rightarrow C$, a rule-based algorithm can infer the relation $A \rightarrow C$ in the test set by using the transitivity property. This makes the relations $A \rightarrow B$, and $B \rightarrow C$, and the relation $A \rightarrow C$ dependent. To break these types of dependence, we have to account for the transitivity property when we randomly split our data. In conclusion, this limitation highlights the need for an algorithm that randomly splits the data while taking into account the transitivity property of prerequisite relations: the Graph Split Algorithm.

CHAPTER 3 Concept Prerequisite Relations as A Strict Partial Order

This chapter describes the AL-CPL dataset used in our experiments and ensures that the mathematical properties of prerequisite relations are held in practice. It also explains the inherent methodological problem of training algorithms for CPL on train and test data that has been split randomly and provides a solution in the form of our Graph Split Algorithm.

3.1 Outtakes for Prerequisite Relations in the AL-CPL dataset

In this section, we show the statistical distribution of the AL-CPL dataset, which is the data we used in our experiments. We also describe and analyze the hypotheses that the creators of this dataset made to identify the prerequisite relations, while drawing conclusions about the appropriate type of model we should use to infer these prerequisite relations. Finally, after establishing the mathematical properties of the prerequisite relation from its definition in Section 2.1, we will test whether these mathematical properties hold in practice for the AL-CPL dataset.

3.1.1 AL-CPL data description

The AL-CPL dataset is based on the manually constructed Wiki Concept Map dataset (Wang et al., 2016). In the Wiki Concept Map dataset, concept pairs are collected from textbooks on different educational domains: data mining, geometry, physics and precalculus. After being extracted, concepts are then manually annotated (Wang et al., 2016).

Then, Liang et al. (2018) asked domain experts to manually correct the labels of the Wiki Concept Map dataset. Due to the asymmetric, and transitive properties of the prerequisite relation between concepts, Liang et al. (2018) were able to add additional non-prerequisite and prerequisite relations to the Wiki Concept Map dataset resulting in the AL-CPL dataset with 1933 unique prerequisite relations between concept pairs labeled 1 and 4442 unique non-prerequisite relations between concept pairs labeled 0 (see Table 3.1).

Finally, Liang et al. (2018) do not include all possible non-prerequisite pairs in the AL-CPL dataset and keep the ratio of non-prerequisite (0) to prerequisite (1) at 30% to avoid extreme class imbalance that would make the classification task harder to solve. In fact, Sinha et al. (2020) showed that the more extreme the class imbalance is for the task, the harder it is to solve the same task using the same model and the same method to counteract this class imbalance. Therefore, it

is best to minimize the class imbalance so that it does not add a superficial difficulty unrelated to solving CPL.

Table 3.1: Statistics on the entire AL-CPL Dataset.

Domain	**# Concepts**	**# Pairs**	**# Prerequisites**
Data Mining	120	826	292
Geometry	89	1681	524
Physics	153	1962	487
Precalculus	224	2060	699
Total (After Merging)	**586 (545 unique)**	**6529 (6375 unique)**	**2002 (1933 unique)**

Because of the emphasis this work puts on improving the performance of methods that solve CPL in the cross-domain setting (see the introduction of Section 2.4 for the definition), we are interested in measuring how similar the domains are to each other. A simple way to do this is to observe the number of concepts that each domain has in common. The higher the number of concepts that two domains share, the more likely they are to have intersecting prerequisite relations, and the easier it will be for the model to predict the remaining prerequisite relations.

Table 3.2 showcases the number of intersecting concepts between the domains. Notably, Precalculus emerges as the most extensive domain, featuring 224 concepts. It intersects with 2 Data Mining concepts, 29 Geometry concepts, and 10 Physics concepts, making it the domain with the biggest number of concepts in common with other domains. Since Data Mining has the fewest intersecting concepts with other domains, we expect it to be the hardest test set for CPL in the cross-domain setting. This turned out to be the case for all of the methods presented in this thesis (see Section 4.3, Table 4.1, cross-domain results).

Table 3.2: Number of Intersecting Concepts per Domain in the AL-CPL dataset.

	Data Mining	**Geometry**	**Physics**	**Precalculus**
Data Mining	120	0	0	2
Geometry	0	89	1	29
Physics	0	1	153	10
Precalculus	2	29	10	224

Further analysis of the dataset's prerequisite relations uncovers intriguing patterns. It is worth noting that despite the extensive coverage of the dataset, a small fraction of concepts remains without any prerequisite relations. Approximately 15% of the 545 unique concepts lack any prerequisite relations. We keep them in the dataset as they are a good measure of the capacity of our algorithms to infer non-prerequisite relations for concepts without any prerequisite example at hand. Keeping them also enables us to compare our results with previous approaches (Miaschi et al., 2019), and (Bai et al., 2021). Additionally, we find that approximately 68.7% of the prerequisite relations are inferred based on the transitivity property of the prerequisite relation. This serves as a motivating factor for the development of our Graph Split Algorithm, as detailed in Section 3.2.

3.1.2 Different graph representations

Additionally, the AL-CPL dataset is provided with a graphical visualization of concept prerequisite relations (Liang et al. 2018). As established in Section 2.1, every set of prerequisite relations between concepts can be represented as a directed acyclic graph (DAG). In this prerequisite graph, each concept is represented as a node and prerequisite relations are represented by directed edges between nodes. For instance, if 'Math' is a prerequisite for 'Precalculus', a directed edge would connect the 'Math' node to the 'Precalculus' node in the prerequisite graph.

As mentioned previously in Section 2.1, graph representations are crucial for solving CPL in this work. Not only do we use graph representations to split the AL-CPL dataset, but we also use them to solve CPL.

Therefore, we use three different types of graphs to represent prerequisite relations between concepts in the AL-CPL dataset based on the different needs of our algorithms. In the following paragraph, we will define the three different graph representations we use in this work: the prerequisite graph without transitive relation (transitive reduction defined by **Definition 11** in Section 2.1), prerequisite graph with transitive relation (transitive closure defined by **Definition 12** in Section 2.1), and the PnP concept graph.

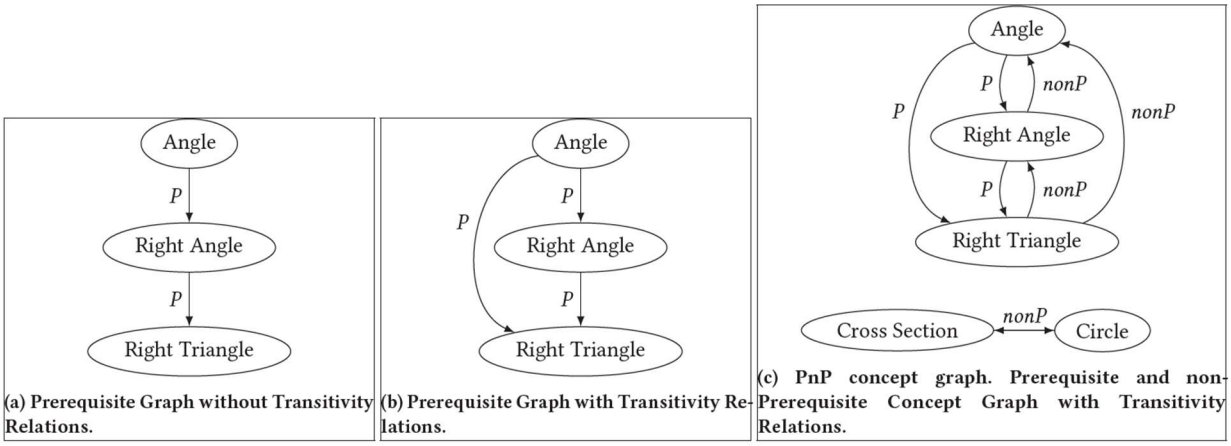


Figure 3.1: Examples of different types of graphs representing concept prerequisite relations. All three are subgraphs of the AL-CPL database. Prerequisite relations have an edge label of $\boldsymbol{P}$ and non-prerequisite relations have an edge label of $\boldsymbol{nonP}$.

Prerequisite graph without transitive relations. Figure 3.1.a shows an example from the AL-CPL dataset of a prerequisite graph without transitive edges, i.e. a graph with only direct edges (see **Definition 13** in Section 2.1). In this graph, each concept is represented as a node and the directed edges connecting them represent prerequisite relations. The edge labels, all set to $\boldsymbol{P}$, indicate the presence of a prerequisite relationship between the connected concepts. This graph representation is generated by computing the transitive reduction of the relations present in the AL-CPL dataset. In practice, we apply the 'nx.transitive_reduction' function from NetworkX to the graph associated with the prerequisite relation present in the AL-CPL dataset which is a DAG (See **Proposition 3** in Section 2.1). Since the graph associated with the prerequisite relations in the AL-CPL dataset is a DAG, the transitive reduction of this graph is unique (see **Definition 13** in Section 2.1). Therefore, because the computation of the transitive closure can be reproduced consistently as the result is unique and deterministic, it can be used by our algorithms without generating an additional source of randomness. Indeed, this graph representation is used by the

Graph Split Algorithm (Algorithm 1 in Section 3.2.2) to randomly select the direct edges and forms the basis of how the algorithm works. Additionally, it is used in our ablation study to assess which graph representation yields the best results for CPL (see section 4.3.2).

Prerequisites graph with transitive relations. Figure 3.1.b presents a prerequisite graph with transitive relations. In addition to the direct edges between the concepts shown in Figure 3.1.a, a new transitive edge labeled $\boldsymbol{P}$ for prerequisite is introduced, connecting `Angle' and `Right Triangle.' This edge is inferred by transitivity using the edges between `Angle` and `Right Angle' and `Right Angle` and `Right Triangle'. This is the graph representation associated with the prerequisite relations mentioned in **Proposition 3**. By adding the transitive edges, the graph captures a more complete view of the prerequisite dependencies between concepts. Due to the transitive relations, the distance between nodes and their root is 1. This implies that our GCNs need fewer hidden layers to represent the relations between all nodes compared to Figure 3.1.a, but this will be discussed further in Section 4.3.1. Similar to the previous graph representation, this graph representation is used by the Graph Split Algorithm and in our ablation study.

Prerequisites and non-prerequisites graph with transitive relations (PnP concept graph). Figure 3.1.c showcases a more complex graph that includes both prerequisite and non-prerequisite relations. It represents a scenario where additional concepts, `Triangle' and `Rectangle', are introduced. The graph incorporates relations induced by transitivity, similar to Figure 3.1.b and incorporates non-prerequisite relations as well. The directed edges between concepts have edge labels of either $\boldsymbol{P}$ or $\boldsymbol{nonP}$, indicating whether the relationship is a prerequisite or non-prerequisite, respectively. This graph representation allows R-GCN to generate learnable representations for all 545 concepts in AL-CPL dataset, including the 83 concepts without any prerequisite relations. This graph representation is used by our model PnPR-GCN which is described in Section 4.1.

3.1.3 AL-CPL founding hypotheses

It is crucial to understand the hypotheses that were made when creating the prerequisite relations in order to understand the nature of the relation.

Wang et al. (2016) built the initial AL-CPL dataset, called the Wiki Concept Map dataset, by making several hypotheses. First, they considered that a pair of concepts to have a prerequisite relationship if they were topically related. They argue that if two concepts cover different

unrelated topics, they are unlikely to have prerequisite relations, they call it the "**Topic-Relatedness**" hypothesis.

Furthermore, Wang et al. (2016) consider that it is unlikely for pairs of concepts with similar topics and similar levels of complexity to have prerequisite relations, such as "isosceles triangle" and "right-angled triangle", they call it the "**Complexity Level Difference**" hypothesis. In their paper, Wang et al. (2016) do not give the explicit equation they use to compute the complexity level difference, but they mention that computations include "Supportive relationship in concept definition [(if A is used in B's definition)], RefD [(which measures the asymmetry of the references between two concepts in their respective Wikipedia pages)], Number of in-links/out-links [of the Wikipedia page associated to the concept], [and] TOC Distance [(distance between the subchapter numbers of the concept pair)]". To summarize, according to Wang et al.'s (2016) "**Complexity Level Difference**" hypothesis, a pair of concepts where A is mentioned in B's definition and vice versa, with the same amount of incoming and outgoing links, a $|RefD|$ close to 0, and both concepts are explained in the same subchapter (for example concept A would be explained in subchapter 3.2.1 and B would be explained in 3.2.2) would imply that they have the same level of complexity, so it would be unlikely for A and B to be linked by a prerequisite relation.

The topic-relatedness hypothesis is a logical hypothesis to make. Intuitively, concepts from unrelated domains should not have prerequisite relations, where the term unrelated domains means that there are no overlapping concepts between the domains. In fact, this hypothesis makes domain knowledge about a concept crucial for identifying prerequisites, thus making topic modeling a task closely related to CPL. Therefore, we take it into account when building our method for solving CPL in Section 4.1.3. On the other hand, the complexity level difference hypothesis reinforces the idea that the prerequisite relation is a partial order on concepts. Indeed, according to the complexity level difference hypothesis, concepts with the same complexity level cannot be compared; therefore, the order defined by the concept prerequisite relation cannot be total (see **Definition 4** in Section 2.1).

Wang et al. (2016) also hypothesize that if concept A is the prerequisite for concept B, then concept A should be introduced before concept B is introduced, which they call the "**Order Coherence**" hypothesis. This is a strong hypothesis to have as it is not always respected in the

educational field in practice. However, in this work, we assume that if concept A is mentioned in the definition of concept B, then concept A has a strong chance to be a prerequisite for concept B. For example, in the AL-CPL dataset, the concept "Polygon" is a prerequisite for "Triangle", and the definition of "Triangle" according to Wikipedia is "A triangle is a **polygon** with three corners and three sides, one of the basic shapes in geometry". This motivates our decision to use concept definitions and contextual embeddings to represent concepts for solving CPL (see Section 4.1.2).

3.1.4 AL-CPL data: a strict partial order on concepts

Then, in the preprocessing stage, Liang et al. (2018) verified that each of the prerequisite relations in the dataset satisfied the required properties of a strict partial order, and asked domain experts to manually correct their labels if necessary. They also extended the dataset using the **irreflexive** and **transitive** properties:

1. They added (B, A) as a negative sample if (A, B) is a positive sample (**irreflexivity**).
2. They added (A, C) as a positive sample if both (A, B) and (B, C) are positive samples (**transitivity**).

Despite their work, we perform our own battery of tests to double-proof the properties of the AL-CPL data.

For each domain prerequisite relations between concepts, we generate the associated directed graph described in **Proposition 3** using the NetworkX Python library: for each concept pair A and B labeled 1 in the dataset, a directed edge (A, B) is added to the graph. We also use NetworkX to run a series of tests to validate whether each domain in the dataset satisfies every property of a strict partial order:

- **Irreflexivity:** we check for the absence of self-loops in the graph. The presence of a self-loop would imply that the relation is not irreflexive, thus our test would throw an error.
- **Asymmetry:** for each directed prerequisite edge (A, B) in the graph, we check whether (B, A) is in the graph. Finding (B, A) is in the graph would imply that the relation is not asymmetric and therefore our test would throw an error.
- **Transitivity:** using `nx.transitive_closure_dag` we generate the transitive closure of the associated graph and we check if the transitive closure and the associated graph have the

same edges. It is important to note that `nx.transitive_closure_dag` fails if the graph has a cycle.

All four domains of the AL-CPL dataset pass the battery of test defined above. Therefore, all four domains of the AL-CPL dataset satisfy the properties of a strict partial order because the graph associated with their prerequisite relations is a DAG without self-loops (**Proposition 4**).

Thus, we conclude that the AL-CPL dataset is an appropriate dataset for training models for CPL.

Finally, as expected in Section 3.1.3, the order described by the AL-CPL data **is not total**. For example, 'Physics' and 'Set' are two concepts from the AL-CPL dataset taken from the physics domain that are not linked by a prerequisite relation. In fact, the pairs (Physics, Set) and (Set, Physics) are labeled as 0 in both directions in the dataset.

3.1.5 Checking for inconsistencies in the AL-CPL dataset

Wary of the possible contradictions in prerequisite relations resulting from merging prerequisite relations from different domains like the counter example of **Proposition 5** where $A \rightarrow B \; and \; B \rightarrow A$, we merge all prerequisite relations from all four domains and check for inconsistencies. We first check for concepts where $A \rightarrow B \; and \; B \rightarrow A$, and we find none. Additionally, we search for inconsistencies in labeling prerequisite pairs between domains. Indeed, when training models on multiple domains, it is crucial that concept pairs with same concept names have the same label otherwise model performance decreases (Fenza et al., 2021).

```
1107  symmetry     geometry              1     geometry
5521  symmetry     geometry              0  precalculus
```

Figure 3.2: Label inconsistency in the AL-CPL dataset.

We find an inconsistency with the labeling of the pair (Symmetry, Geometry) in both Geometry and Precalculus domains. In the prerequisite pairs associated to the domain of Geometry, the concept of 'Geometry' is labeled as a prerequisite for the concept 'Symmetry'. Whereas in the Precalculus domain, the pair is labeled as non-prerequisites. In this work, we decide that Geometry is a prerequisite for Symmetry based on the topic relatedness and order coherence hypotheses introduced in Subsection 3.1.1 and our own assessment of prerequisite relations.

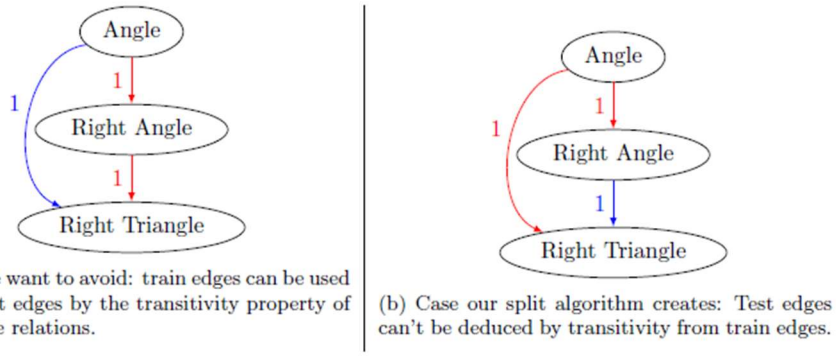
3.2 Graph Split Algorithm

By examining the concept prerequisite graph generated from the AL-CPL dataset, we noticed that some prerequisite relations are easier to deduce than others due to the transitive property (see Figure 3.3). In practical scenarios, a model that can infer prerequisites with limited information is more valuable than one that relies on transitivity, since a rule-based algorithm can easily make inferences based on this property.

This insight led us to develop a graph-based random split algorithm we call the Graph Split Algorithm to avoid that scenario, while maintaining the dataset's transitive and direct edge ratio (see Algorithm 1). In this context, transitive edges are those that are deductible through the transitivity property, whereas direct edges are not.

3.2.1 The problem with a standard random split algorithm

In ML, randomly splitting the data for a classification task is a standard practice in order to train algorithms. It is by far the most popular method and is mainly used to split a dataset into k folds for cross-validation (James et al., 2013). By randomly splitting the data into a training set and a test set, the distribution in the test set is statistically similar to the training set. This is what we call a random split of the data. This enables models to be trained in a similar setting as the test set while measuring the capacity to generalize on the test set and avoiding overfitting.



(a) Case we want to avoid: train edges can be used to infer test edges by the transitivity property of prerequisite relations.

(b) Case our split algorithm creates: Test edges can't be deduced by transitivity from train edges.

Figure 3.3: Two possible results of splitting train and test edges of an AL-CPL subgraph. Train edges are in red, test edges are in blue.

However, the random split approach in ML can be problematic as it assumes that data points are independent of one another. Indeed, there are cases where the data has dependencies due to specific characteristics or structures that random splitting cannot account for:

- **Spatial Autocorrelation in data:** Ploton et al. (2020) show that using a traditional random split to validate methods leads to an overoptimistic assessment of model predictive power for spatially correlated data. In fact, when spatial validation methods are used, the same methods exhibit almost no predictive power. To evaluate the impact of spatial autocorrelation in the data, the article uses a spatial K-fold cross-validation approach. This involves dividing the data into spatial clusters, with each cluster alternately serving as training or test sets. Through this method, the model undergoes testing on data from spatial clusters without being exposed to nearby data points, thus limiting the information leakage between spatially correlated data points.
- **Information leakage from grouped data samples:** Drawing on my experience in computer vision from a previous internship, I encountered a distinct form of information leakage between training and testing datasets. We worked with a dataset comprising 1000 images, distributed across 50 object categories. Each category consisted of 10 images of the same object, captured under varying lighting conditions and angles, representing 100 unique objects. Our objective was to train a classifier that could generalize across different lighting and angles. However, upon examining the feature maps of our Convolutional Neural Network (CNN), we discovered that the network had learned a shortcut for object identification. Specifically, it recognized unique defects like tears on objects, which it then erroneously applied to similar defects in the test set. This flaw became apparent when the model failed to correctly identify new objects from the same categories. To address this, we revised our data splitting approach to ensure that images of the same object were assigned exclusively to either the training or test set. This change effectively mitigated the problem of the model using the wrong cues for correct identification and improved its ability to generalize effectively.

Table 3.3: Average percentage of inferable edges in the test set using the transitive closure of the training graph for a random split and a graph split (see Algorithm 1).

	Data Mining	**Geometry**	**Physics**	**Precalculus**	**AVG**
Random Split	43.5%	71.0%	51.2%	53.2%	54.7%
Graph Split	0.0%	0.0%	0.0%	0.0%	0.0%

In our study, we identified a unique form of information leakage linked to the transitivity property of concept prerequisite relations. For instance, if 'Angle' is a prerequisite for 'Right Angle', and 'Right Angle' is a prerequisite for 'Right Triangle', then by transitivity, 'Angle' becomes a prerequisite for 'Right Triangle'. This specific scenario, as illustrated in Figure 3.3 (left side), occurs on average for 54.7% of the total edges in the test set generated by a random split as shown in Table 3.3. This leads us to hypothesize that due to the transitivity property, model assessment is optimistic in previous papers. In other words, this information leakage due to the transitivity property leads to an optimistic assessment of model predictive power, similarly to the scenario cited above. In Section 3.2.2, we will elaborate on our strategy for addressing this particular form of information leakage.

3.2.2 The Pseudo-code

The idea behind Algorithm 1 is simple: exclude direct edges from the training set if they can be used to deduce transitive edges in the test set. As illustrated in Figure 3.3, if the edge between `Angle' and `Right Angle' was in the test set, then `Angle` → `Right Triangle` would not be deducible by transitivity. To implement this, Algorithm 1 randomly selects subsets of direct and transitive edges (the latter being deducible from the former) from the entire dataset to include them in the test set. Here is the pseudo-code for our Graph Split Algorithm:

Algorithm 1: Graph Split Algorithm

1. Initialize $\tilde{G}_{\mathrm{Pd}}$ as the copy of the prerequisite graph without transitive edges G_{Pd}, E_t and E_d as empty lists storing respectively transitive and direct edges, and nbreDirectTest as the number of direct edges required in the test set.

2. // In the while loop we split the Prerequisite Edges into two categories: direct edges and transitive edges.

3. **While** len(E_d) < nbreDirectTest **do**

4. $\tilde{G}_{Pd}$, RemovedEdge ← RemoveRandomEdgeFromGraph($\tilde{G}_{Pd}$)

5. ConceptA, ConceptB ← RemovedEdge[0], RemovedEdge[1]

6. anc ← GetAncestorsOfNode(PrereqG, ConceptA)

7. desc ← GetDescendantsOfNode(PrereqG, ConceptB)

8. TransitiveEdgesType1 ← All Prerequisite Edges between Ancestors **anc** and Descendants **desc**

9. TransitiveEdgesType2 ← All Prerequisite Edges between Descendants **desc** and RemovedEdge[1]

10. TransitiveEdgesType3 ← All Prerequisite Edges between RemovedEdge[0] and Descendants **desc**

11. E_t ← Concatenate E_t with TransitiveEdgesType1, TransitiveEdgesType2, TransitiveEdgesType3

12. E_d ← Concatenate E_d with RemovedEdge

13. **End While**

14. // Then, we select Direct Edges (E_d) and a portion of the Transitive Edges (E_t) while making sure that they are **not inferable by transitivity** from edges in the train set and place them in the **test set**. We make sure that after the split, the ratio of **Direct Edges** and **Transitive Edges** remains the same in the train and test split.

This Graph Split Algorithm has a nice property, it generates splits where the test edges are not inferable by the transitivity property using edges in the train set. Let us define this property and prove that Algorithm 1 satisfies it:

Definition 17 (A non-inferable-by-transitivity test split). *Let* S_{Train} and S_{Test} be a split of the data S. *We say that* S_{Test} *is non-inferable-by-transitivity when the intersection of the transitive closure of the train set with the test set is empty:* $S^{+}_{Train} \cap S_{Test} = \emptyset$.

Theorem 1. *All test splits generated by Algorithm 1 are non-inferable-by-transitivity.*

Proof:

Let S_{Train} and S_{Test} be a split resulting from the Graph Split Algorithm (Algorithm 1) which has terminated successfully.

If S_{Train} and S_{Test} are a result of Algorithm 1, it means that all thirteen steps described in the pseudo-code were successfully executed. And, according to line 14, we have $S^{+}_{Train} \cap S_{Test} = \emptyset$.

Q.E.D.

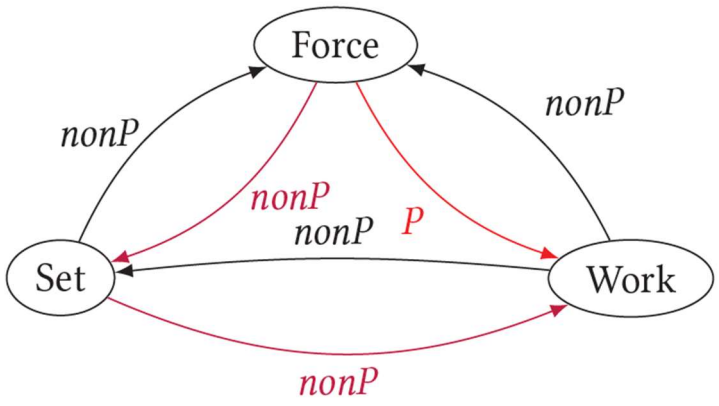


Figure 3.4: An example showcasing that non-prerequisite relations are non-transitive taken from the Physics concept prerequisite graph of the AL-CPL database.

In our approach, we specifically applied the Graph Split Algorithm to prerequisite relations, acknowledging their transitive nature. However, we intentionally refrained from applying this split to non-prerequisite relations, given that they do not exhibit transitivity in the context of a

partial order among concepts. As per **Definition 4**, a total order requires that any two concepts must be ordered. In Figure 3.4, 'Force' and 'Set' are two unordered concepts, where 'Force' and 'Set' and 'Set' and 'Work' are linked by non-prerequisite edges in both directions. Nevertheless, the direct prerequisite relation between 'Force' and 'Work' confirms that non-prerequisite relations do not follow transitivity. This distinction is crucial in our methodology, ensuring the algorithm's application is tailored to the structure of the concept prerequisite relations.

3.2.3 The diamond edge case

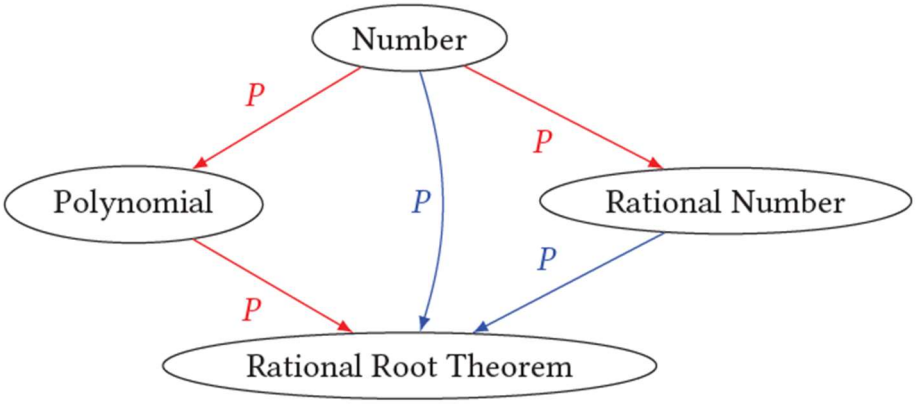


Figure 3.5: Example taken from the precalculus domain showing a directed diamond graph. Train edges are in red, test edges are in blue.

According to graph theory (*Graph Theory*, 2008), a diamond graph is an undirected graph with 4 vertices and 5 edges. We extend this definition to directed graphs, which makes the graph in Figure 3.5 a directed diamond graph. The problem with these types of graphs is that even though we remove the directed edge that was used to infer the transitive edge, there is still another direct edge that can be used to infer that transitive edge. Figure 3.5 illustrates that with the direct edges between `Number`, `Polynomial` and `Rational Root Theorem`. Although the direct edge between `Rational Number` and `Rational Root Theorem` is in the test set, direct edges between `Number`, `Polynomial` and `Rational Root Theorem` can be used to infer the transitive edge `Number` and `Rational Root Theorem`. However, by making sure that we select transitive edges that cannot be inferred by transitivity from train edges in line 13 of Algorithm 1, we avoid this edge case in practice.

Table 3.4: Number of Diamond Edge Cases per Domain in the AL-CPL dataset.

	Data Mining	**Geometry**	**Physics**	**Precalculus**
Number of Diamond Graphs	3	11	16	24

In Table 3.4, we observe that diamond graphs are present for each domain in the AL-CPL dataset.

For the AL-CPL dataset, line 14 in Algorithm 1 is sufficient to take care of this edge case which can be observed by our result in our GitHub repository [4]and that is shown in the last row of Table 3.3: 0% of test edges are inferable by transitivity from train edges.

3.2.4 Line 14 of Algorithm 1 and Proof of Correctness

Indeed, in the line 14 of the pseudocode of Algorithm 1 the statement "we select […] the Transitive Edges (E_t) while making sure that they are **not inferable by transitivity** from edges in the train set and place them in the **test set"** is added to account for the edge case described in the Section 3.2.3. However, it does not provide clear instructions on how we implement the solution to the information leak caused by the transitivity property (see Section 3.2.1).

Therefore, the aim of this section is to provide the Python code for the specific instructions that achieve what is claimed in line 14, to formally prove that the Python code is correct, and to explain how the code functions. To prove that the algorithm is "correct" is to prove it selects a non-inferable-by-transitivity test split as defined in **Definition 17** Section 3.2.2.

Figure 3.6 illustrates the Python code that is responsible for selecting the edges that cannot be inferable by transitivity from the edges in the train set. In this code, `train_ds` contains all the prerequisite relations between concepts for a given domain in the AL-CPL dataset. At this point in the algorithm, we did not seperate the train data from the test data. `direct_rel` holds the direct edges randomly selected, and `trans_rel_notinf_test_candidates` holds the corresponding transitive edges (see Algorithm 1).

We call `PrereqG_copy` the potential training graph. Precisely, `PrereqG_copy` is a prerequisites graph with transitive relations (see Section 3.1.2). `PrereqG_copy` contains all the prerequisite

[4] https://github.com/Lama-West/PnPR-GCN_ACM_SAC_24.git.

relations in `train_ds` minus the direct edges stored in `direct_rel` and the transitive edges stored in `trans_rel_notinf_test_candidates` that are both candidate edges to be selected for the test set.

The function `GetCurrentTrainingGraph` builds the current training graph by taking all prerequisite relations in `train_ds` and removing the direct edges, `direct_rel`, and the transitive edges, `trans_rel_notinf_test_candidates`. Given a set of transitive edges T and a graph $G = (E, V)$, the function `RemoveInferableEdges` removes from the set of transitive edges all edges present in the transitive closure of the said graph and returns T'. The resulting set of transitive edges T' has this property: $T' \cap E^+ = \emptyset$.

```
1.    PrereqG_copy = GetCurrentTrainingGraph(train_ds, direct_rel,
                          trans_rel_notinf_test_candidates)
2.    new_trans_rel_notinf_test_candidates =
                    RemoveInfereableEdges(trans_rel_notinf_test_candidates,
                                          PrereqG_copy).copy()
3.    while(new_trans_rel_notinf_test_candidates.shape[0] !=
            trans_rel_notinf_test_candidates.shape[0]):
            #hill climbing on a convex problem, will terminate,
                  but can terminate with an empty set
4.          trans_rel_notinf_test_candidates =
                  new_trans_rel_notinf_test_candidates
5.          PrereqG_copy = GetCurrentTrainingGraph(train_ds, direct_rel,
                               trans_rel_notinf_test_candidates)
6.          new_trans_rel_notinf_test_candidates =
                          RemoveInfereableEdges(trans_rel_notinf_test_candidates,
                                                PrereqG_copy).copy()
```

Figure 3.6: Python code that selects the transitive edges that are non-inferable-by-transitivity as defined in **Definition 17**.

In the Python code in Figure 3.6, we iteratively build the training graph based on the removed direct and transitive edges. We then select transitive edges that are non-inferable-by-transitivity as defined in **Definition 17** and place them in the test set. The test set is the union of the removed direct and transitive edges. If no edges are removed from the set of transitive edges, then the training graph and the test graph satisfy the property defined in **Definition 17**. Moreover, the algorithm terminates because there is only a finite number of transitive edges that can be inferred through transitivity. We provide a complete and formal proof of correctness below for more clarity.

Proof of correctness:

We are going to prove that with the given assumptions when the code in Figure 3.6 terminates, the post condition "test edges are non-inferable-by-transitivity as defined in **Definition 17** Section 3.2.2" is True. And we are going to prove that termination indeed occurs.

Assumptions (also valid for Algorithm 1):

- Assumption 1: `PrereqG_copy` is a DAG, ensuring that there is only one way of splitting the edges of the graph between direct edges, E_d, and transitive edges, E_t, where $E_d \cap E_t = \emptyset$ (see **Definition 13** in Section 2.1).
- Assumption 2: `PrereqG_copy` edge set is finite, ensuring termination.

Loop Invariant (LI) formulation:

Transitive edges in `new_trans_rel_notinf_test_candidates` cannot be inferred by transitivity by using edges in `PrereqG_copy`, where `PrereqG_copy` is the entire prerequisite graph excluding the direct edges `direct_rel` and the transitive edges `trans_rel_notinf_test_candidates`. And `new_trans_rel_notinf_test_candidates.shape[0]` $\leq$ `trans_rel_notinf_test_candidates.shape[0]`.

Initialization: before entering the loop, does our LI hold?

Before entering the loop, lines 1 and 2 of the code in Figure 3.6 ensure that the LI holds. Indeed, line 1 defines the `PrereqG_copy` as the entire prerequisite graph minus the direct edges `direct_rel` and the transitive edges `trans_rel_notinf_test_candidates`. Then in line 2, edges in `new_trans_rel_notinf_test_candidates` are computed using the `RemoveInferableEdges` function applied to `PrereqG_copy` and `trans_rel_notinf_test_candidates`. By definition of this function, these edges cannot be inferred by transitivity by using edges in `PrereqG_copy`: $\text{PrereqG}^{+}_{\text{copy}} \cap \text{trans\_rel\_notinf\_test\_candidates} = \emptyset$ (property of the `RemoveInferableEdges` function). Finally, since by construction `new_trans_rel_notinf_test_candidates` has fewer or the same amount of edges compared to `trans_rel_notinf_test_candidates`, we get `new_trans_rel_notinf_test_candidates.shape[0]` $\leq$ `trans_rel_notinf_test_candidates.shape[0]`.

In conclusion, LI is satisfied before entering the loop.

Maintenance: does our loop invariant hold after each loop iteration?

We use inductive reasoning to prove the maintenance of our LI. We assume that the loop invariant is true upon entering the loop. We then execute each line of code in the loop and prove that the LI holds after every iteration.

In Figure 3.6, line 4 updates the value of `trans_rel_notinf_test_candidates` to current value of ` new_trans_rel_notinf_test_candidates`, the statement `new_trans_rel_notinf_test_candidates.shape[0]` ≤ `trans_rel_notinf_test_candidates.shape[0]` holds. Line 5 and 6 are similar to line 1 and 2. Therefore, `new_trans_rel_notinf_test_candidates` cannot be inferred by transitivity by using edges in `PrereqG_copy` (line 6), where `PrereqG_copy` is the entire prerequisite graph excluding the direct edges `direct_rel` and the transitive edges `trans_rel_notinf_test_candidates` (line 5).

In conclusion, our LI holds after every iteration.

Partial correctness: is the postcondition true when the loop halts?

When the loop stops, the negation of the condition of the while loop (exit condition) is satisfied: `new_trans_rel_notinf_test_candidates.shape[0]` = `trans_rel_notinf_test_candidates.shape[0]`. This implies that no edges were removed from `trans_rel_notinf_test_candidates`, thus we have `trans_rel_notinf_test_candidates` = `new_trans_rel_notinf_test_candidates`. Additionally, LI holds after the last iteration (maintenance). We conclude that the edges of `trans_rel_notinf_test_candidates` cannot be inferred by transitivity by using edges in `PrereqG_copy`, where at the end of the loop `PrereqG_copy` is the entire prerequisite graph without the direct edges `direct_rel` and `trans_rel_notinf_test_candidates`. By defining and selecting test edges as $S_{Test} = \text{trans\_rel\_notinf\_test\_candidates} \cup \text{direct\_rel}$ and the training edges as $S_{Train} = \text{PrereqG}_{copy}$, we get $S_{Test} \cap S^{+}_{Train} = \emptyset$.

Here, $\text{direct\_rel} \cap S^{+}_{Train} = \emptyset$ because of Assumption 1. Indeed, lets prove this by using the property of DAGs. Because direct_rel edges are removed from `PrereqG_copy`, we have $\text{direct\_rel} \cap S_{Train} = \emptyset$. Or, since `PrereqG_copy` is a DAG (Assumption 1), edges cannot be both transitive and direct: $E_{Direct} \cap E_{Transitive} = \emptyset$. Therefore, the intersection between the transitive edges of the training graph, $S^{+}_{Train} \backslash S_{Train}$, and the removed direct edges, direct_rel, is empty: $\text{direct\_rel} \cap (S^{+}_{Train} \backslash S_{Train}) = \emptyset$. Hence, we have $\text{direct\_rel} \cap S^{+}_{Train} = \emptyset$.

In conclusion, the post-condition "test edges are non-inferable-by-transitivity as defined in **Definition 17** Section 3.2.2" is True if the algorithm terminates.

Termination: does our algorithm terminate?

To prove the loop terminates, we start by finding an expression $e = \text{trans\_rel\_notinf\_test\_candidates.shape}[0]$, where e is a natural number on entering the loop and after every iteration. The value of e is used in the loop's exit condition. Because $e \geq 0$, if we prove that e is strictly decreasing for each iteration, then we prove that the code in Figure 3.6 terminates.

If we enter the loop, then the condition of the while loop is satisfied, hence `new_trans_rel_notinf_test_candidates.shape[0]` $< e$. Line 4 updates the value of e as the value of `new_trans_rel_notinf_test_candidates.shape[0]` which is strictly inferior to e. Therefore, the new value of e, e', is strictly inferior to e: $0 \leq e' < e$ and e strictly decreases for each iteration. When we take into account the LI and the fact that the number of edges in a set is positive, we get the following inequation: $0 \leq \text{new\_trans\_rel\_notinf\_test\_candidates.shape}[0] \leq e$. Since e is strictly decreasing after each iteration, we conclude that our algorithm terminates in the outermost N iterations, where N is the number of edges in `trans_rel_notinf_test_candidates` before entering the loop which is finite due to Assumption 2. In fact, since e is a natural number, it can decrease by one at each iteration minimum. Therefore, it would take outermost N iterations to reach 0. And when it does, we have $0 \leq \text{new\_trans\_rel\_notinf\_test\_candidates.shape}[0] \leq e$ and $e = 0$, which implies that $\text{new\_trans\_rel\_notinf\_test\_candidates.shape}[0] = e = 0$ and the loop termination condition is satisfied.

In summary, the algorithm terminates after the outermost N iterations where N is finite.

This concludes our proof of correctness, since we have proven that the Python code in Figure 3.6 terminates and that the post-condition is True.

Q.E.D.

Remarks:

- *If `PrereqG_copy` is a tree, the while loop in Figure 3.6 is not entered (the diamond edge case is avoided).*

- *We have no theoretical guarantees that the Python code in Figure 3.6 avoids the trivial solution of the equation $T' \cap E^+ = \emptyset$ which is the empty set: $T' = \emptyset$. In practice, however, we find all the transitive edges we need in all four domains for the test set (see Section 3.2.5).*

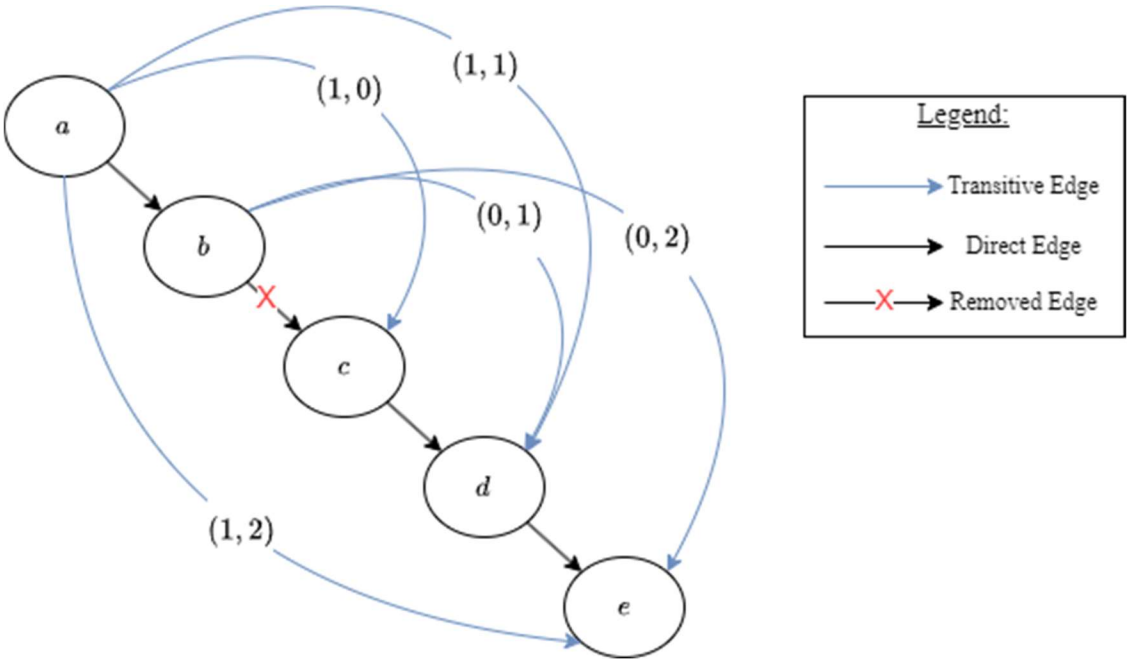


Figure 3.7: Example of a DAG with direct (black) and transitive (blue) edges. Each transitive edge is annotated according to Equation 3.1.

Figure 3.6 shows a part of the code responsible for choosing a non-inferable-by-transitivity test split. It is part of a bigger loop, that iteratively adds the transitive edges found in the while loop of Algorithm 1 to the test set. In this loop, the order of which we add transitive edges to the test set affects how many edges will be inferable by transitivity by using edges in the train set.

Indeed, in Figure 3.7 we removed the direct edge (b, c) to place it in the test set. Consequently, we must select transitive edges to place in the test set so that the ratio of transitive to direct edges is maintained in both the test and training sets. If we add the edge (b, e) first, we notice that it can be inferred by using the edges (b, d) and (d, e) which are both still in the training set. On the other hand, if we add (a, c) first to the test set, because (b, c) has already been removed we cannot infer this edge by transitivity by using the remaining edges in the train set. This is because the combined distance of the nodes of the transitive edge (a, c) to the nodes of the removed edge

(b, c) is less than that of (b, e). In a sense (a, c) is "closer" to (b, c) than (b, e) is. To formalize this intuition and implement this strategy in our algorithm, we define a labeling system for the transitive edges with respect to the removed edge in Equation 3.1 and define a preorder based on these labels in Equation 3.2.

For each transitive edge, we compute a pair of integers following this formula:

$$\begin{pmatrix} ShortestPathLength(TransitiveEdge[0], ConceptA), \\ ShortestPathLength(ConceptB, TransitiveEdge[1]) \end{pmatrix}. \quad (3.1)$$

Where $ShortestPathLength$ is the function that computes the length of the shortest path between two nodes in the graph G_{Pd}, and G_{Pd} is the prerequisite graph without transitive edges, only direct edges (see Section 3.1.2). $TransitiveEdge$ is a transitive edge found in the while loop of Algorithm 1 that contains a pair of nodes in G_{Pd} where $TransitiveEdge[0] \rightarrow TransitiveEdge[1]$ and $TransitiveEdge[0]$ is a prerequisite for $TransitiveEdge[1]$. The rest of the notations are similar to the notations in Section 3.2.2 and in Algorithm 1.

We order the transitive edges with respect to the preorder we define below:

$$\forall i, j, k, l \in \mathbb{N}, (i, j) \leq (k, l) \Leftrightarrow i + j \leq k + l. \quad (3.2)$$

We call the relation defined in Equation 3.2 a preorder because it is a transitive and a reflexive relation, but it is not antisymmetric. Indeed, we have $(1, 0) \leq (0, 1)$ and $(0, 1) \leq (1, 0)$, but $(0, 1) \neq (1, 0)$. However, how we order pairs of equal sums does not affect the non-inferable-by-transitivity property of the test split because they cannot be used to infer each other using the transitivity property, so the order in which we add them to the test set can be arbitrary.

Finally, we follow the preorder defined in Equation 3.2 when adding transitive edges to the test set. To illustrate, if we were to add the transitive edges in Figure 3.7 to the test set, we would add them in this order $(1, 0)$, $(0, 1)$, $(1, 1)$, $(0, 2)$, $(1, 2)$. Thus, by combining this strategy of adding transitive edges and the code in Figure 3.6 that ensures that the postcondition "test edges are non-inferable-by-transitivity as defined in **Definition 17** Section 3.2.2" is True, we get the complete pseudo-code of line 14 in Algorithm 1 (see Algorithm 2).

Algorithm 2: Pseudo-code of Line 14 in Algorithm 1.

1. Initialize k to 0, trans_rel_notinf_test_candidates to an empty list, and nbreDirectTest as the number of direct edges required in the test set.

2. **While** len(trans_rel_notinf_test_candidates) $<$ nbreTransTest **and** $k <$ len(trans_rel) **do**

3. trans_rel_notinf_test_candidates $\leftarrow$ Add the k-th ordered transitive edge set in trans_rel to trans_rel_notinf_test_candidates and remove the transitive edges that are inferable by transitivity using the edges of the current training graph

4. **If** len(trans_rel_notinf_test_candidates) $>$ nbreTransTest **then**

5. Remove randomly len(trans_rel_notinf_test_candidates)-nbreDirectTest edges from the last transitive edges added to trans_rel_notinf_test_candidates

6. **End If**

7. trans_rel_notinf_test_candidates $\leftarrow$ The result of the code in Figure 3.6 applied to trans_rel_notinf_test_candidates

8. $k \leftarrow k + 1$

9. **End While**

Here, trans_rel is the dictionary of transitive edges found by the while loop in Algorithm 1 where the keys are the labels of the transitive edges defined in Equation 3.1 and the values are the transitive edges associated to the label. Transitive edges are ordered according to the preorder defined in Equation 3.2 using the keys of trans_rel. And nbreTransTest is the number of transitive edges required for the test set defined in Equation 3.3:

$$\text{nbreTransTest} = \frac{|\text{E}_{\text{Transitive}}|}{|\text{E}|} \times |S_{Test}|. \tag{3.3}$$

Where $|S_{Test}|$ is the number of required prerequisite edges in the test set, $|\text{E}|$ is the total number of prerequisite edges in the domain-specific AL-CPL dataset, and $|\text{E}_{\text{Transitive}}|$ is the number of transitive edges in the domain-specific AL-CPL dataset. Adding nbreTransTest transitive edges to the test set ensures that the ratio of **Direct Edges** and **Transitive Edges** remains the same in

the test split with respect to the initial ratio of the prerequisite edges belonging to the domain-specific AL-CPL dataset.

Moreover, Algorithm 2 is correct because it is partially correct, and it terminates. It is partially correct because the final test edges in line 7 are selected by the code in Figure 3.6, which is correct. And, under the same assumptions as the code in Figure 3.6, Algorithm 2 terminates because len(trans_rel) is finite (Assumption 1), k is bounded $0 \leq k < \text{len(trans\_rel)}$, and k is strictly increasing after each iteration. Therefore, Algorithm 2, which implements line 14 of Algorithm 1, is correct.

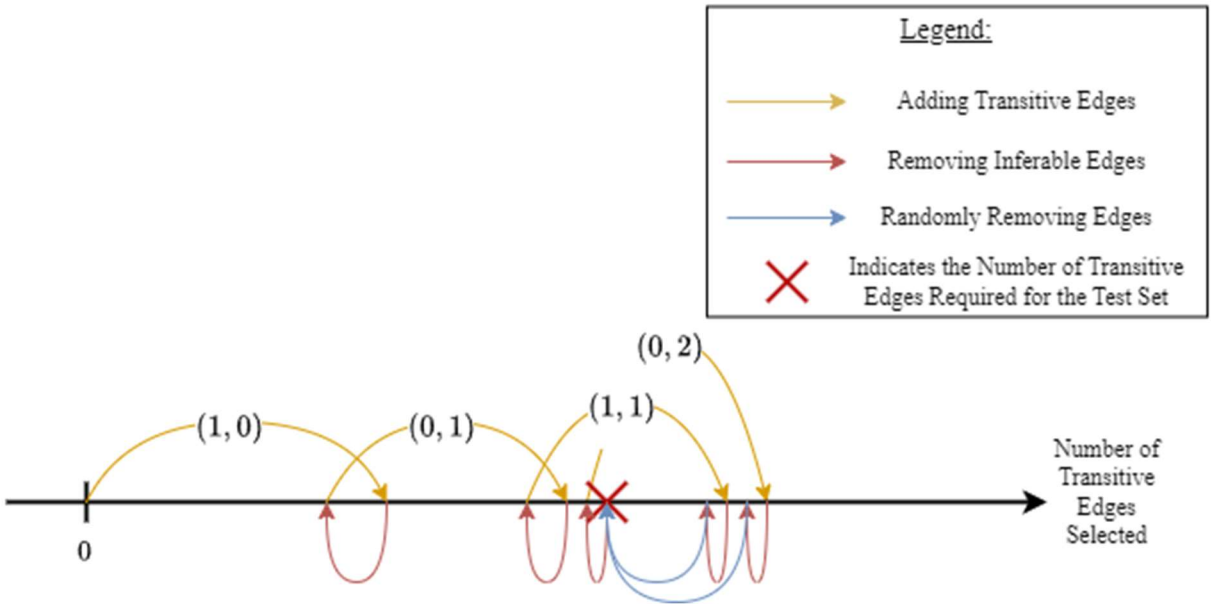


Figure 3.8: Illustration of the variation of the number of transitive edges selected for the test set during the execution of Algorithm 2.

To illustrate how Algorithm 2 works, we execute it on an example similar to Figure 3.7 but with more edges, and show the evolution of the number of transitive edges selected for the test set (see Figure 3.8):

- In the **first iteration**, line 3 of Algorithm 2 added the transitive edges labeled $(1, 0)$ to trans_rel_notinf_test_candidates (first yellow arrow in Figure 3.8). The if block in line 4 is not entered because there are not enough transitive edges for it to meet the condition. Then line 7 removed the transitive edges inferable by transitivity (first red arrow in Figure 3.8).
- At the **second iteration** edges labeled $(0, 1)$ were added similarly to the first iteration.

- For the third iteration, edges labeled $(1, 1)$ were added. Because the number of transitive edges in trans_rel_notinf_test_candidates is larger than nbreDirectTest, the condition is met to enter the if block in line 4. Thus, the statements in line 5 are executed, randomly removing len(trans_rel_notinf_test_candidates)-nbreDirectTest from the previously added transitive edges to trans_rel_notinf_test_candidates (blue arrow linked to the $(1, 1)$ yellow edge in Figure 3.8). Notice the red arrow linking the blue arrow linking the $(1, 1)$ yellow edge in Figure 3.8. It is due to line 3 also removing inferable edges by transitivity. Additionally, line 7 removes edges which prevent the algorithm from terminating just yet.
- This triggers the **fourth and final iteration** upon which $(0, 2)$ edges were added. At the end of the fourth iteration, the number of transitive edges required for the test set is met and the algorithm terminates.

In conclusion, in this section we ultimately proved that our implementation of line 14 in Algorithm 1 is correct under the assumption that `PrereqG_copy` is a finite DAG.

3.2.5 Impact of the Graph Split Algorithm on the Training Prerequisite Graphs

The aim of this section is to find any anomalies in the prerequisite graphs generated by Algorithm 1. Therefore, this section investigates how the structural properties of the prerequisite training graphs generated by the Graph Split Algorithm (Algorithm 1) differ from the structural properties of the AL-CPL original prerequisite graphs. Where the original prerequisite graph of a domain refers to the prerequisite graph with transitive relations (see Section 3.1.2) associated with the prerequisite relations of a domain in the AL-CPL dataset before it is split into a training and a test set. And the prerequisite training graph refers to the prerequisite graph with transitive relations associated with the prerequisite relations in a training set of a domain. There are four domains in the AL-CPL, resulting in four original prerequisite graphs, and thus four columns of statistics in Table 3.5, Table 3.6, Table 3.7, and Table 3.8.

There are two methods for randomly splitting the data between a train and a test set that are being evaluated in this section: the random split used by SOTA and previous work, which involves randomly splitting the data into k folds for cross-validation, and our proposed method which is the Graph Split Algorithm (refer to Algorithm 1). The Graph Split Algorithm eliminates the information leak caused by the random split (refer to Section 3.2.1 and 3.2.3). We use $k = 5$ as in previous work, to ensure a fair comparison of our experimental results to theirs. This means that

for each split method and domain, there are five different prerequisite training graphs. The mean and the standard deviation ($\mu \pm \sigma$) for each statistic presented in Table 3.5, Table 3.6, Table 3.7, and Table 3.8 are included for a more detailed statistical description.

For this investigation, we compare the prerequisite training graphs generated by both of the split methods mentioned previously and the original graphs of the AL-CPL dataset. We are specifically interested in the prerequisite training graphs and not the test graphs generated by the splits, because if the original prerequisite graph and the prerequisite training graph are completely different, the model will be biased. Specifically, we observe how the prerequisite training graphs and the original prerequisite graph of AL-CPL differ in terms of number of unique concepts (nodes of the graph) with prerequisite relations in Table 3.5, the average distance between reachable nodes in Table 3.6, the average node indegree[5] in Table 3.7, and the number of weakly connected components (refer to **Definition 14** in Section 2.1) in Table 3.8.

These structural features are important to consider because they affect how the model is trained and the choice of its optimal hyperparameters (refer to Section 4.3.1). Particularly, the average distance between reachable nodes and the number of weakly connected components provide insight into which method to use for agglomerating nodes to generate graph-based node embeddings. Indeed, the most effective approach for generating graph-based node embeddings differs between graphs with multiple connected components and a relatively low average distance between reachable nodes, and those with few connected components and a relatively large average distance between reachable nodes (Chiang et al., 2019; Hamilton et al., 2018). Refer to Section 2.3 for an explanation of how GCNs generate graph-based node embeddings.

Notably, the number of prerequisite and non-prerequisite relations for each prerequisite training graph is not shown in this section because it is a constant in both splitting methods, computed based on the percentage of edges needed for the split. There will be no variation ($\sigma = 0$), so it does not provide any information about the difference between the prerequisite training graphs.

Table 3.5 is the first of the four tables presented in this section. It shows the number of unique concepts with prerequisite relations in the original AL-CPL domain graphs and in the resulting prerequisite training graph for each domain.

---

[5] Whether we take the indegree or the outdegree the results will be the same: $\frac{|E|}{|V|}$. Where $|E|$ is the number of edges and $|V|$ is the number of nodes in the graph.

Table 3.5: Number of unique concepts **with prerequisite relations** in the AL-CPL dataset before and after splitting the data into 5 training sets for each domain. The mean and the standard deviation over the 5 training sets are displayed in this format where applicable: $\mu \pm \sigma$.

*Explanation for why these numbers differ from the ones in Table 3.1 is underlined in the next paragraph.

Graph Type	Data Mining	Geometry	Physics	Precalculus
Original AL-CPL Graphs	90*	88*	124*	196*
Graph Split Train Graphs	83.4 ± 1.2	85.6 ± 0.49	116.6 ± 2.245	182.4 ± 1.625
Random Split Train Graphs	87.6 ± 1.356	87 ± 1.095	120.2 ± 1.939	191 ± 2.417

Our first observation related to the results in Table 3.5 is that the number of concepts is not the same as what was shown in Table 3.1. This is because concepts without prerequisites are left out of the count in Table 3.5. It is important to note that these concepts are still present in the prerequisite training graph as nodes, but they are not linked by any prerequisite relations to other concepts. However, they can still be linked by non-prerequisite relations in the training set (refer to PnP concept graph in Section 3.1.2). The reason we do not account for nodes without prerequisite relations in this section is because both the Graph Split Algorithm and the random split behave similarly towards these nodes.

Our second observation related to Table 3.5 is that on average the prerequisite training graphs generated by Graph Split Algorithm have fewer nodes with prerequisite relations. This is due to the way Algorithm 1 selects the prerequisite edges. Indeed, Algorithm 1 randomly selects a direct edge from the original prerequisite graph along with a portion of transitive edges associated with the nodes of the selected direct edge. This increases the chance of one of the nodes being left out without any prerequisite relations.

In conclusion, there were no anomalies detected regarding the number of unique concepts with prerequisite relations between prerequisite training graphs generated by the Graph Split Algorithm and the random split.

Table 3.6 shows the average node distance between reachable nodes in the original AL-CPL domain graphs and in the resulting prerequisite training graph for each domain (see **Definition 10** Section 2.1 for the definition of reachability). Since prerequisite relations define a partial order on concepts, the original prerequisite training graph is equal to its own transitive closure. By definition, given a node v in this graph, there exists a direct link between v and all its reachable nodes. Therefore, the **distance** between v and all its reachable nodes in the **original prerequisite graph** is 1.

Table 3.6: Average distance between reachable nodes in the AL-CPL dataset before and after splitting the data into 5 training sets for each domain. The mean and the standard deviation over the 5 training sets are displayed in this format where applicable: $\mu \pm \sigma$.

*Explanation for the constant 1 is underlined in the previous paragraph.

Graph Type	Data Mining	Geometry	Physics	Precalculus
Original AL-CPL Graphs	1*	1*	1*	1*
Graph Split Train Graphs	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.002 ± 0.000
Random Split Train Graphs	1.104 ± 0.013	1.151 ± 0.017	1.114 ± 0.018	1.119 ± 0.009

Our first observation from Table 3.6 is that the random split modifies the average distance between nodes compared to the original graph. This is because it is possible for the random split to remove transitive edges linking two nodes without removing the direct edges in between the two nodes, causing their distance to be increased by one node.

In contrast and by design, Algorithm 1 removes both the direct edges between nodes and the transitive edges that can be inferred by the direct edges, thereby making nodes in the graph either unreachable or directly connected to other nodes. This makes the 1.000 ± 0.000 average distance

between nodes a predictable effect. Also, it has a positive side effect: the prerequisite training graphs resulting from the Graph Split Algorithm is more similar to the original graphs in terms of average distance between reachable nodes compared to the prerequisite training graphs resulting from the random split.

However, for Precalculus, there are cases where two nodes remain reachable by a distance of two nodes, even though their direct link and transitive links are removed from training. This is due to the diamond edge case (see previous Section 3.2.3). The diamond subgraphs make it possible to have two or more direct edge paths between nodes, which explains the 1.002 ± 0.000 result. Indeed, Precalculus is by far the set with the most diamond edge cases (24, refer to Table 3.4) and has the greatest number of prerequisites.

In conclusion, it is the Graph Split Algorithm that generates the prerequisite training graphs that are more similar to the original graphs in terms of the average distance between reachable nodes.

Table 3.7: Average node indegree in the AL-CPL dataset before and after splitting the data into 5 training sets for each domain. The mean and the standard deviation over the 5 training sets are displayed in this format where applicable: $\mu \pm \sigma$.

Graph Type	Data Mining	Geometry	Physics	Precalculus
Original AL-CPL Graphs	3.244	5.955	3.927	3.566
Graph Split Train Graphs	2.808 ± 0.040	4.907 ± 0.028	3.346 ± 0.065	3.076 ± 0.027
Random Split Train Graphs	2.667 ± 0.041	4.818 ± 0.079	3.242 ± 0.096	2.926 ± 0.054

According to Table 3.7, both splits decrease the average indegree of nodes, which comes with no surprise as it is a direct cause of removing edges from the original prerequisite graph and putting them in the test set. The random split and graph split have similar statistics for all domains. However, the statistics of the Graph split train are consistently closer to the original AL-CPL dataset.

Table 3.8: Number of weakly connected components in the AL-CPL dataset before and after splitting the data into 5 training sets for each domain. The mean and the standard deviation over the 5 training sets are displayed in this format where applicable: $\mu \pm \sigma$.

Graph Type	Data Mining	Geometry	Physics	Precalculus
Original AL-CPL Graphs	3	2	2	3
Graph Split Train Graphs	3.4 ± 0.490	2.2 ± 0.400	2.8 ± 0.748	4.4 ± 0.800
Random Split Train Graphs	3.6 ± 0.490	1.8 ± 0.400	2.4 ± 1.020	3.0 ± 0.632

The main takeaway from Table 3.8 is that the Graph Split Algorithm tends to create more weakly connected components than the random split. On average, the difference between the number of weakly connected components of the prerequisite training graphs generated by the Graph Split Algorithm (GS) and the Random Split (RS) is $\frac{1}{4}\sum_{i=1}^{4} WCC_{GS_i} - WCC_{RS_i} = 0.5$. Once again, this is an expected outcome from the Graph Split Algorithm. Indeed, it causes the paths between weakly connected components to be more likely to break compared to a regular random split.

In summary, the Graph Split Algorithm has its expected impact on the prerequisite training graphs and behaves in some cases like the random split. However, overall, we conclude that it is the graph split that generates the prerequisite training graph that is the most accurate to the real-world scenario. After extensive checks, no anomalies were detected; therefore, we deem this split suitable for training transductive ML methods for CPL. Section 4.2.1 details how the random split and the Graph Split Algorithm are used to train and evaluate the performance of our models. Additionally, Section 4.3 includes the results of our models on both split types.

CHAPTER 4 Sentence Transformers and Graph Convolutional Networks: A Fitting Framework for Concept Prerequisite Learning

This chapter presents the approach and the methodology we designed to solve Concept Prerequisite Learning using Natural Language Processing and Machine Learning tools.

First, Section 4.1 explains the methodology of our approach to solving CPL. Then Section 4.2 proceeds to describe in detail our experimental. Finally, Section 4.3 presents the results of our experiments.

4.1 Our Approach with Graph Convolutional Networks

In Section 2.1, we proved that the prerequisite relation is a strict partial order on concepts; therefore, that we can represent concepts and their prerequisite relations as a DAG. Because graphs naturally represent relationships and structure between nodes, we chose to use graph-based algorithms for the prediction of prerequisite relations. Additionally, we opt for R-GCN as our graph-based algorithm because it has shown impressive results on graphs with different link types (Schlichtkrull et al., 2017), and in our case we have prerequisite and non-prerequisite relations.

However, R-GCNs require node representations as input to operate (refer to Section 2.3). Therefore, we use textual information and the Sentence Transformers library (Reimers & Gurevych, 2019) to generate concept embeddings for our node representations, as it has already been proven that textual-based concept embeddings improve the performance of R-GCN-based architectures for CPL (Li et al., 2020; Tang et al., 2023; Zhang et al., 2022).

Thus, our method uses two sets of inputs to solve CPL as a binary classification problem (refer to Section 2.3.1): *concept embeddings* and the *graph structure*. This section covers three main points: the graph structure in Section 4.1.1, concept embeddings in Section 4.1.2, and our classification architecture in Section 4.1.3.

4.1.1 The Prerequisite and Non-prerequisite Concept Graph

Formal Graph Definition. We define the prerequisite and non-prerequisite concept graph $G = (V, E)$ as the union of the prerequisite graph $G_P = (V, E_P)$ and the non-prerequisite $G_{nonP} = (V, E_{nonP})$, where V is a set of $N = |V|$ vertices representing concepts, and $E =$

$\{(v_i, P, v_j)|(v_i, v_j) \in E_P\} \cup \{(v_i, nonP, v_j)|(v_i, v_j) \in E_{nonP}\}$ is the edge set. A_P is the binary adjacency matrix, where $A_P[i,j] = 1$ if there is a prerequisite relation between concept i and concept j. A_{nonP} is the binary adjacency matrix, where $A_{nonP}[i,j] = 1$ means there is a non-prerequisite relation between concept i and concept j.

We use the PnP concept graph as input (see Subfigure c in Figure 3.1) because it allows GCNs to incorporate both the concepts without prerequisite relations and the concepts with prerequisite relations into their graph neighborhood-based embedding generation process. This cannot be achieved by the prior two graphs depicted in Subfigure a and b in Figure 3.1.

4.1.2 Concept Representation using Contextual Embeddings

In this section, we explain how we generate our concept embeddings using the definitions of concepts and the Sentence Transformer library[6]. The concept embeddings can be found on our GitHub page[7].

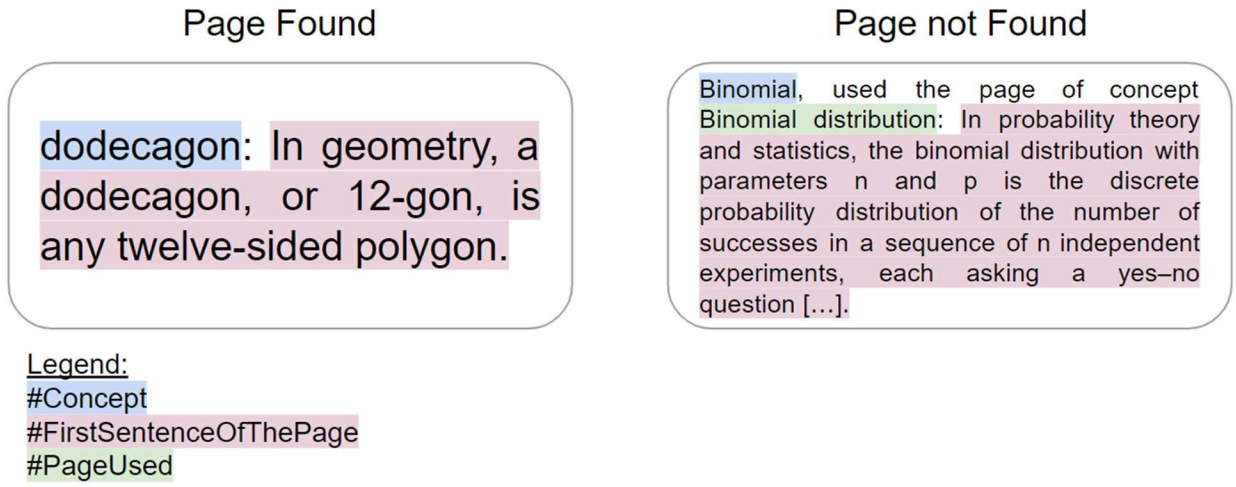


Figure 4.1: Example of the sentence used when we find the Wikipedia page associated with the concept (left), and a sentence when we use another page related to the concept (right).

One of the main differences with previous approaches solving CPL is that we use the Sentence Transformers library (Reimers & Gurevych, 2019) to generate concept embeddings for CPL, because previous work misused transformers in the context of CPL (refer to Section 2.4.3).

[6] https://www.sbert.net/index.html
[7] Available at https://github.com/Lama-West/PnPR-GCN_ACM_SAC_24.git.

To do so, we first need to find a defining sentence for each concept. We query the Wikipedia API and retrieve the first sentence of the Wikipedia page associated with the concept. Next, we form a sentence similar to a definition and create a sentence with this structure: `#Concept: #FirstSentenceOfThePage.' (`Page Found' example on the **left** of Figure 4.1).

We chose the first sentence of the page because, according to Wikipedia's manual of style, it should define the subject with simple words.

We found the corresponding Wikipedia page for 537 of the 545 concepts (98.53%) in the AL-CPL dataset. If we don't find the Wikipedia page associated with the concept, we use the first sentence of the first page suggested by the Wikipedia API. The resulting sentence is structured as follows: `#Concept, used the page of concept #PageUsed: #FirstSentenceOfThePage.' (`Page not Found' example on the **right** of Figure 4.1).

SMPNet Embeddings. Each sentence defining a concept (see Figure 4.1) is fed to `all-mpnet-base-v2'. We chose `all-mpnet-base-v2'[8], which is pre-trained using Sentence-BERT's (Reimers & Gurevych, 2019) procedure on MPNet (Song et al., 2020), because it has the best performance on sentence embedding tasks of all pre-trained model available of the Sentence Transformers library. We also tested other pre-trained models from the SentenceTransformers library on our CPL task, and `all-mpnet-base-v2' consistently outperformed them. The dimension of the embeddings generated using `all-mpnet-base-v2' (SMPNet) is $d_{\text{SMPNet}} = 768$ per concept.

4.1.3 PnPR-GCN: our R-GCN-based Classifier Model

In this section, we present our R-GCN-based architecture for CPL (see Figure 4.2) and will refer to it as PnPR-GCN.

[8] Available at https://huggingface.co/sentence-transformers/all-mpnet-base-v2.

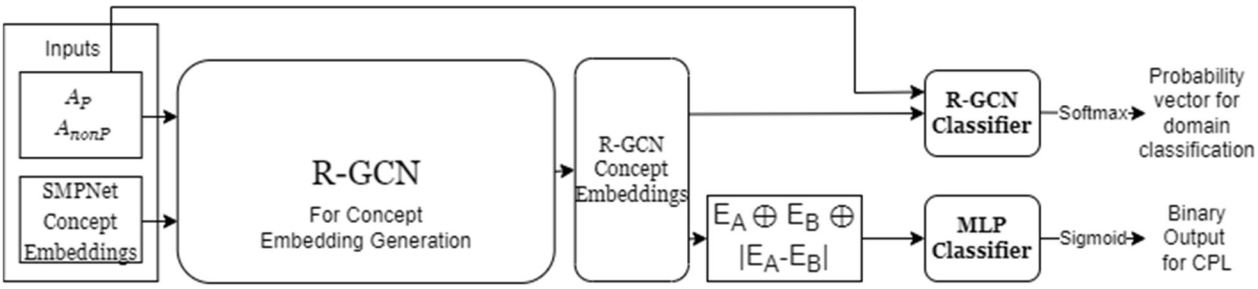


Figure 4.2: PnPR-GCN. Our GCN-based architecture that uses a Multi-Layer Perceptrons (MLP) classifier for CPL and another one-layer R-GCN for concept domain classification.

The first R-GCN block of our architecture computes concept embeddings based on the adjacency matrices A_P and A_{nonP}, and the SMPNet concept embeddings serving as the initial node representations (refer to Section 2.3.2 for more details). A_P and A_{nonP} are derived from the PnP concept graph representing the ACL-PL train set (refer to Section 3.1.2). The concept embeddings computed by the R-GCN are called `R-GCN Concept embeddings' in Figure 4.2. Using R-GCNs to embed concepts enables us to distinguish between prerequisite and non-prerequisite relations. Precisely, for the R-GCN, the weight matrix that multiplies A_P is different from the weight matrix that multiplies A_{nonP}, whereas a regular GCN has only one trainable weight matrix (Kipf & Welling, 2017; Schlichtkrull et al., 2017). We then use the concept embeddings generated by the R-GCN block for two tasks in parallel: CPL and concept domain classification. Where domain classification is formalized as a K multiclass classification problem, where $K \in \mathbb{N}$ represents the number of different domains/topics in the dataset.

Our idea of solving CPL and concept domain classification simultaneously stems from the need to handle concepts from the domain whose prerequisite relations have been held out of the training set, also known as the cross-domain setting. Indeed, if we only train our model on CPL, concepts without any prerequisite or non-prerequisite relations in the train set will maintain the random initialization of their associated R-GCN weight matrices, even post-training. This scenario occurs for the majority of the concepts in the test set of the cross-domain setting, i.e. concepts that do not have prerequisite relations in more than one domain (see Table 3.1). Additionally, according to the "**Topic-Relatedness**" hypothesis made by the authors of the AL-CPL dataset and described in Section 3.1.1, concepts that are not topically related cannot be

prerequisites. Therefore, training the model on the concept domain classification task should improve the model's accuracy in identifying prerequisite relations between concepts.

Therefore, because GCNs perform particularly well on node classification tasks (Kipf & Welling, 2017; Schlichtkrull et al., 2017), we introduced an additional task into our training process: domain classification for all concepts. Concept domain classification is a multiclass classification task that infers the topic to which a concept belongs, such as 'Physics' or 'Statistics'. By learning the domain classification task, all concepts have updated weight matrices that are learned during the training phase, even concepts without any prerequisite or non-prerequisite relations. The strategy of effectively learning multiple tasks with a single network, along with the selection of appropriate loss hyperparameters, draws inspiration from existing work on Convolutional Neural Networks (CNNs) (Kuga et al., 2017) and Cycle-GAN (Zhu et al., 2020). Notably, CNNs trained on multiple tasks simultaneously yield superior latent representations compared to single-task CNNs (Kuga et al., 2017). That being said, training our architecture sequentially, first on the concept domain classification task and then CPL, was a viable option, but we chose to learn both in parallel to circumvent the catastrophic forgetting problem with neural networks (Kirkpatrick et al., 2017). To our knowledge, we are the first to propose a GCN-based architecture that does link prediction and node classification at the same time for CPL. For the definitions of node classification and link prediction, see Section 2.3.

In the following, we present the CPL and the concept domain classification tasks in addition to their loss functions.

CPL Objective

Upon acquiring the updated concept embeddings $z_a, z_b \in \mathbb{R}^{n_d}$ of dimension n_d from the R-GCN for the concept pair (a, b), we concatenate both embeddings. The concatenation of embeddings, z_a, z_b, and their absolute difference $|z_a - z_b|$ is represented by $E_A \oplus E_B \oplus |E_A - E_B|$ in Figure 4.2. We feed $(z_a, z_b, |z_a - z_b|)$ to a Multi-Layer Perceptron (MLP), G_p, with a sigmoid output $\sigma\left(G_p(z_a, z_b, |z_a - z_b|)\right) = \hat{y}_{ab} \in (0,1)$ to predict the true label $y_{ab} \in \{0,1\}$. $\sigma: x \mapsto \frac{1}{1+e^{-x}}$ is the sigmoid function and $\hat{y}_{ab}$ is the binary output for CPL depicted in **Error! Reference source not found.**.

Similar to previous work, we use the binary cross-entropy loss as the prerequisite loss. Due to the imbalanced nature of the dataset, where the prerequisite relations are the underrepresented class, we used a weighted binary cross-entropy in some experiments. We compute the positive class weight as:

$$w_1 = \frac{n_0}{n_1}, \tag{4.1}$$

where $n_0 \in \mathbb{N}$ is the number of non-prerequisite relations (negative examples) and $n_1 \in \mathbb{N}$ is the number of prerequisite relations (positive examples). By using class weights, we handle class imbalance and ensure that the model pays equal attention to both classes. If the loss is not rebalanced, $w_1 = 1$ in Equation 4.1.

Thus, the MLP classifier G_p tries to minimize the objective function below:

$$\mathcal{L}_{Prereq}(G_p) = -\left(\frac{1}{n_{train}}\right)\sum_{y_{ab}}[\, w_1 \cdot y_{ab} \cdot log(\hat{y}_{ab}) + (1 - y_{ab}) \cdot log(1 - \hat{y}_{ab})] \,. \tag{4.2}$$

Where n_{train} is the number of pairs in the training set, w_1 is the positive class weight defined in Equation 4.1, $\hat{y}_{ab} \in (0,1)$ is the prediction of the MLP G_p, and $y_{ab} \in \{0,1\}$ is the true label of the relation between concept a and b.

Concept Domain Classification Objective

Let $z_a \in \mathbb{R}^{n_d}$ be the updated concept embedding for concept a generated by R-GCN. And let $d_a \in \{0,1\}^K$ be the one-hot encoded domain of concept a. Because concept domain classification is a node classification task, we can leverage the graph structure by using an additional single layer R-GCN block G_d as a domain classifier (Schlichtkrull et al., 2017). The probability vector for domain classification is computed as such: $softmax\left(G_d(z_a, A_p, A_{nonP})\right) = \hat{d}_n \in [0,1]^K$, which is the probability vector for domain classification depicted in Figure 4.2 that tries to predict the true domain label d_a of concept a. Consequently, we minimize the following cross-entropy loss on all labeled nodes:

$$\mathcal{L}_{Domain}(G_d) = -\frac{1}{|V|}\sum_{n \in V}\left[\sum_{i=1}^{K} d_{ni}\log(\widehat{d}_{ni})\right]. \tag{4.3}$$

Where $|V|$ is the number of nodes in the graph, $\widehat{d}_n \in [0,1]^K$ is the R-GCN block prediction for the true domain label $d_n \in \{0,1\}^K$ of concept n, and $d_{ni}, \hat{d}_{ni}$ are the i-th value of vectors d_n and $\widehat{d}_n$ respectively.

The GCN architecture in Figure 4.2 is transductive, meaning all nodes are present in the training set. Only the edges are split between the training set and the test set. In sum, our PnPR-GCN classifier tries to minimize a weighted sum of both losses in Equation 4.2 and 4.3. The weights of each loss $\lambda, \mu \in \mathbb{R}$ are chosen via hyperparameter search on the validation set. Thus, the PnPR-GCN architecture tries to minimize the objective function below:

$$\mathcal{L}_{GCN}(G_p, G_d) = \lambda \mathcal{L}_{Prereq}(G_p) + \mu \mathcal{L}_{Domain}(G_d). \tag{4.4}$$

Where $(\lambda, \mu) \in \mathbb{R}^2$, G_d is the domain classifier and G_p the CPL classifier.

Differentiability of PnPR-GCN

Equation 4.4, presented in the previous section, shows the loss of our PnPR-GCN model $\mathcal{L}_{GCN}$. $\mathcal{L}_{GCN}$ is a sum of two differentiable losses $\mathcal{L}_{Prereq}$ and $\mathcal{L}_{Domain}$ which makes $\mathcal{L}_{GCN}$ itself differentiable. For the sake of clarity, we are going to compute an explicit version of the loss using the chain rule. Without loss of generality, we are going to consider every neural network layer in PnPR-GCN to be a fully connected layer (refer to Section 2.2.1 for definition). The proof remains the same if you swap fully connected layers with R-GCN layers, as both outputs are differentiable, but we do it for the sake of simplicity.

Therefore, we will assimilate PnPR-GCN as a sequence of fully connected layers (see Figure 4.3) and compute its loss with respect to the weights A_{embed} of the first fully connected layer FC_{embed} in the sequence using the chain rule.

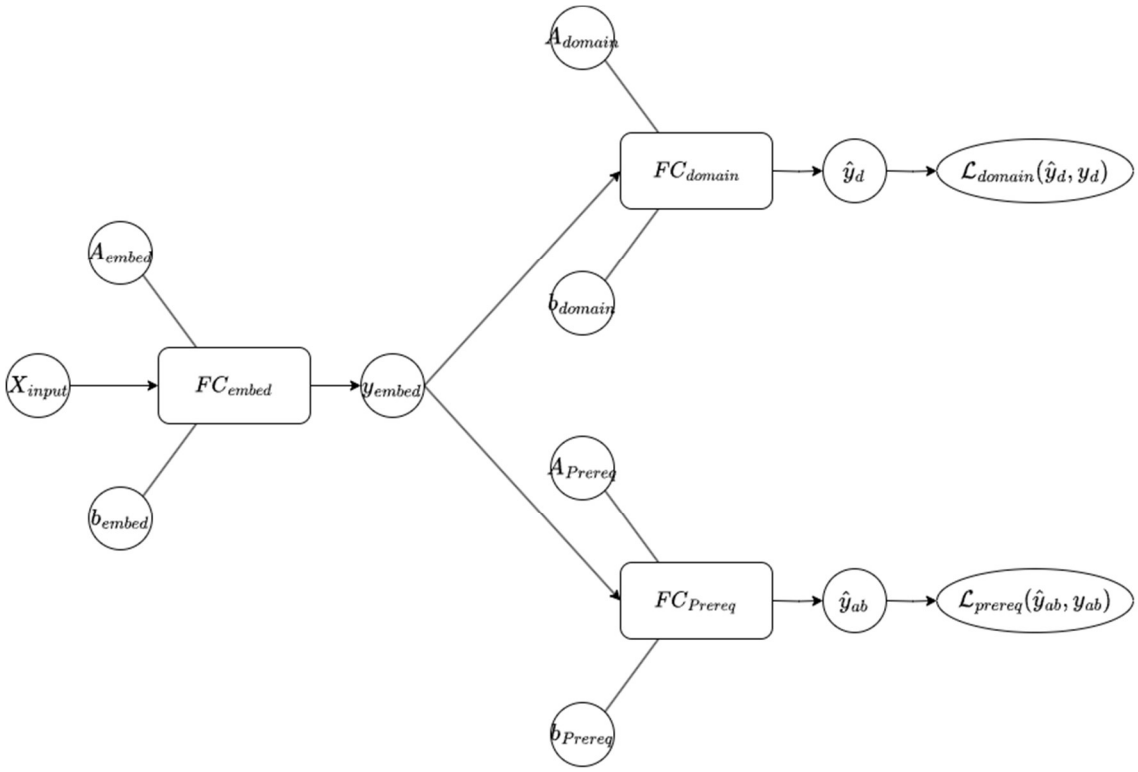


Figure 4.3: PnPR-GCN representation with Fully Connected layers.

To achieve this, we represent the R-GCN block that creates the embeddings as a fully connected layer called FC_{embed}, the R-GCN block that does the domain classification as FC_{domain} and the MLP classifier for prerequisite learning as FC_{Prereq} (see Figure 4.2 for original PnPR-GCN and Figure 4.3 for the simplified version with fully connected layers).

Let's compute the partial derivative of $\mathcal{L}_{GCN}$ with respect to A_{embed} using the chain rule:

$$\begin{aligned}\frac{\partial \mathcal{L}_{GCN}}{\partial A_{embed}} &= \lambda \frac{\partial \mathcal{L}_{Prereq}(\hat{y}_{ab}, y_{ab})}{\partial A_{embed}} + \mu \frac{\partial \mathcal{L}_{Domain}(\hat{y}_d, y_d)}{\partial A_{embed}} \\ &= \lambda \frac{\partial y_{embed}}{\partial A_{embed}} \times \frac{\partial \hat{y}_{ab}}{\partial y_{embed}} \times \frac{\partial \mathcal{L}_{Prereq}(\hat{y}_{ab}, y_{ab})}{\partial \hat{y}_{ab}} \\ &\quad + \mu \frac{\partial y_{embed}}{\partial A_{embed}} \times \frac{\partial \hat{y}_d}{\partial y_{embed}} \times \frac{\partial \mathcal{L}_{Domain}(\hat{y}_d, y_d)}{\partial \hat{y}_d}. \end{aligned} \tag{4.5}$$

Let's suppose the weights of the neural network are updated according to a Stochastic Gradient Descent (SGD), then A_{embed} update equation would be:

$$A_{embed}^{[new]} = A_{embed}^{[old]} - \alpha \frac{\partial \mathcal{L}_{GCN}}{\partial A_{embed}} . \quad (4.6)$$

Where $\alpha \in \mathbb{R}$ is the learning rate, $A_{embed}^{[new]}$ the updated value of A_{embed} and $A_{embed}^{[old]}$ the old value.

By combining Equation 4.5 and 4.6 we conclude not only that each layer of PnPR-GCN can learn by backpropagation, but also that the weights of the embedding layers are updated proportionally to λ times the prerequisite loss $\mathcal{L}_{Prereq}$ and μ times the domain loss $\mathcal{L}_{Domain}$. This gives us an intuition of the impact of the hyperparameters λ and μ on the training of the model, which helps in our hyperparameter grid search.

4.2 Experimental Setting

To demonstrate the efficiency of our proposed methods, we test our approaches on the AL-CPL dataset in the in-domain setting and in the cross-domain setting for CPL, defined in Section 3.1, and compare our results with Bai et al. (2021) and Miaschi et al. (2019). For each train, validation, and test set, we run experiments three times and record the average. How we generate these dataset splits is described in the following Section 4.2.1.

As previously stated in Section 4.1.3, we employed a re-balanced version of the cross-entropy loss for improved outcomes (see Equation 4.1 and 4.2), opting against oversampling, unlike the SOTA approach (Miaschi et al., 2019).

The code of our Graph Split Algorithm, the data splits used for our experiments, and the concept embeddings we generated can be found on our GitHub page[9].

4.2.1 Dataset Split Methodology

This section aims to explain how we split the AL-CPL dataset into a training, validation and test split for our experiments.

[9] https://github.com/Lama-West/PnPR-GCN_ACM_SAC_24.

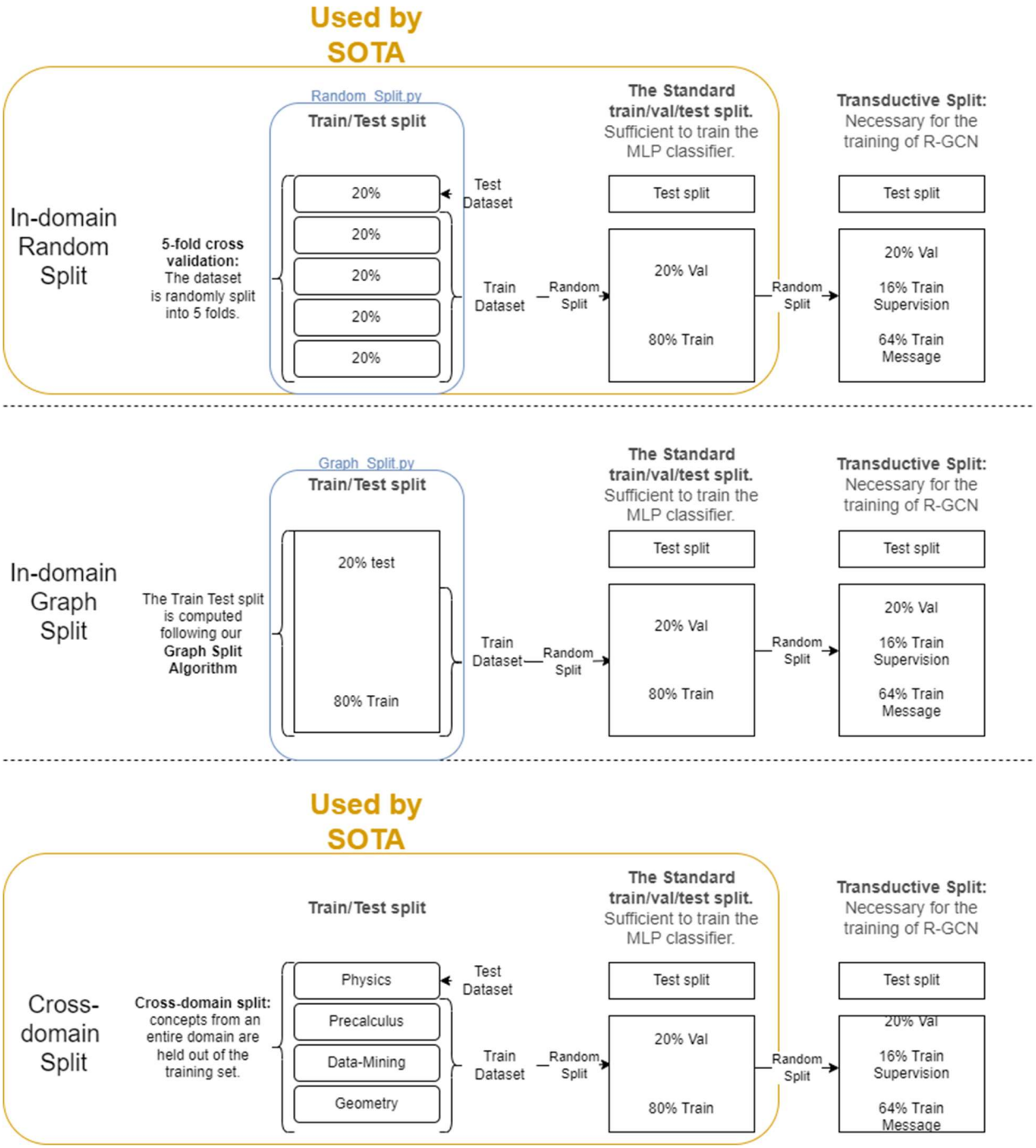


Figure 4.4: Illustration of the steps we followed to split our data for the traditional classifiers and the GCN based classifiers. Three different methodologies are showed here: the In-domain Random Split, the In-domain Graph Split, and the Cross-domain split.

Figure 4.4 illustrates the steps of our three different strategies for splitting our data for the traditional ML classifiers and the GCN based classifiers.

In Figure 4.4 'Random Split' refers to the randomized data split mentioned in Section 3.2.1 obtained from running a function akin to `sklearn.model_selection.train_test_split`, while the "Graph Split" pertains to our Graph Split Algorithm's featured in Algorithm 1 (see Section 3.2.2). Additionally, the definitions of the in-domain and cross-domain setting can be found in Section 2.4 and is specific to CPL. In the in-domain setting, both "Graph Split" and "Random Split" generate 5 distinct training, validation, and test splits for each domain. Finally, the transductive split is the method of splitting the training data that is recommended by Leskovec for transductive link prediction with GCNs (refer to Section 2.3.1). In the following paragraphs, we are going to explain the three different methodologies illustrated in Figure 4.4:

1. **In-domain Random Split:** In order to compare results with prior research on AL-CPL (Bai et al., 2021; Liang, Ye, Wang, et al., 2018; Miaschi et al., 2019), we adopt their methodology of splitting the data. Therefore, we split the prerequisite and non-prerequisite edges from one domain of the AL-CPL dataset into 5 folds for cross-validation using the random split as described in Section 3.2.1. In this setting, models must be trained and tested on concept pairs belonging to the same domain. This is used for in-domain validation and involves a random split of the data. It's further divided into:
    a. **5-fold cross-validation:** The dataset is randomly divided into five equal parts, or "folds." This process is repeated five times (once for each fold), and each time a different fold is selected to be used as the test set while the remaining four folds serve for training and validation sets.
    b. **The Standard training, validation, and test Split:** The four remaining folds are further subdivided by randomly selecting 80% of the prerequisite and non-prerequisite edges in the four remaining folds and placing them in the training set and the rest in the validation set. At this stage, we have three subdivisions: a training set with 64% of the total number of the prerequisite and non-prerequisite edges from one domain of the AL-CPL dataset, a validation set with 16%, and a test set with 20%.
    c. **Transductive Split:** This split is necessary for models that use graph-based methods like R-GCNs (refer to Section 2.3.1). The training set is additionally subdivided so that 20% is used for supervision, and the remaining 80% is used as a training message. In sum, 51.2% of the total number of the prerequisite and non-

prerequisite edges from one domain of the AL-CPL dataset is used for train message, 12.8% is used for train supervision, 16% is used for validation and 20% is used for the test set.

2. **In-domain Graph Split:** Very similar to the previously described in-domain setting, but instead of using the regular random algorithm to split the data into a train and test split, we use our graph split (see Algorithm 1). This method is designed to prevent the data leakage from the transitive property of prerequisite relations described in Section 3.2.1. It involves:
    a. **Graph Split Algorithm:** The first step of this approach is to split the prerequisite and non-prerequisite edges following our Graph Split Algorithm, leading to 80% of the prerequisite and non-prerequisite edges being used for training and 20% for testing.
    b. The rest of the sub-methodologies under this category are the same as described above in the in-domain random split.
3. **Cross-domain Split:** This type of split is designed to test generalization across different domains:
    a. **Cross-domain split:** For cross-domain tests, previous research adopted a leave-one-domain-out strategy (Miaschi et al., 2019), training classifiers on three domains and testing on the remaining one. This approach is efficient for gauging model responses to unfamiliar domains. However, it is worth noting that domains, while distinct, can have overlapping concepts (refer to Table 3.2). The extent of this overlap can influence model performance. In addition, due to the few intersecting concepts between domains (see Table 3.2), the case we are trying to avoid, which is test edges being inferable by transitivity from train edges, should be rare. Therefore, we decided not to apply the Graph Split Algorithm (Algorithm 1) to split the training and test set for the cross-domain setting.
    b. The rest of the sub-methodologies under this category are the same as described above in the in-domain random split.

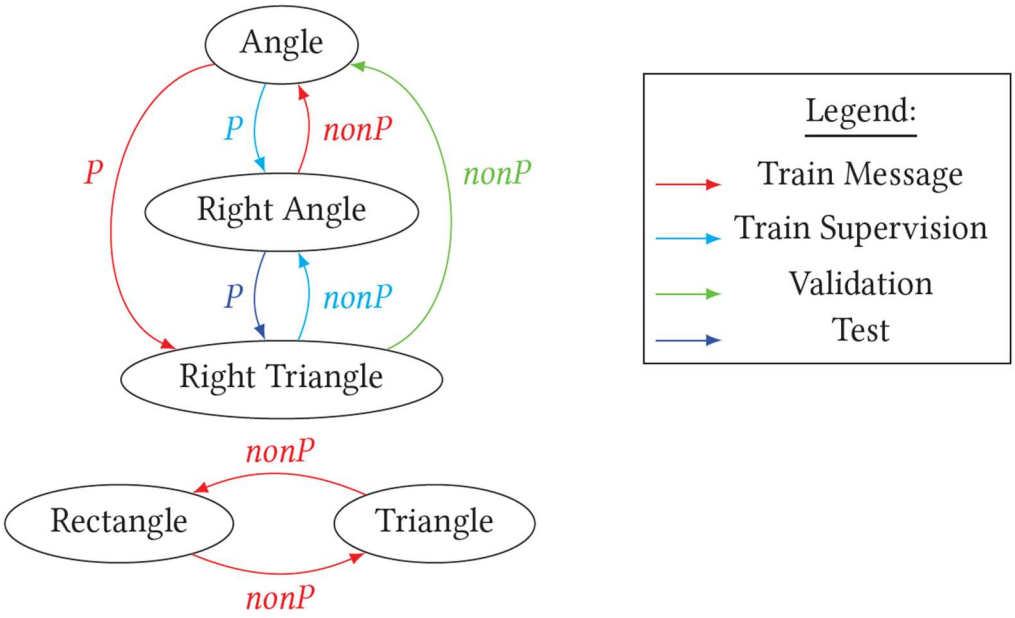


Figure 4.5: An example of a transductive split for link prediction on a subgraph of the AL-CPL dataset.

Indeed, given the transductive nature of GCNs, we implement the transductive split for link prediction (refer to Section 2.3.1)[10]. This split is only used when training our PnPR-GCN architecture and builds upon the earlier train and test splits in both in-domain and cross-domain settings. All nodes are visible across all dataset partitions. however, because edges play roles in both message passing and supervision, we must reserve certain edges for supervision during the training, validation, and testing stages. An example of a transductive split for link prediction of the AL-CPL dataset is shown in Figure 4.5.

In the next section, we are going to detail our choice of baselines for our experiments on the AL-CPL dataset.

4.2.2 Baselines

Concept Embedding Baselines

We present the following methods of representing concepts as a baseline for our own method of representing concepts:

[10] The code we used to reproduce the transductive split is available here https://zqfang.github.io/2021-08-12-graph-linkpredict/ (we used option 2 with 'torch_geometric').

Random Baseline Embeddings. To quantify how much textual-based embeddings impact the performance of our R-GCN-based architecture, we compare model results with random embeddings. Random embeddings are generated following a random uniform distribution $U([0,1])$. The dimension of the random embeddings is $d_{Random} = 300$ for each concept, to match the dimension of the FastText Embeddings.

FastText Embeddings. The name of concepts are represented using FastText embeddings (Bojanowski et al., 2017), averaging normalized representations for multi-word concepts. However, this method can misrepresent concepts like `Machine Learning' due to individual word meanings. Despite its limitations, FastText remains a robust word embedding baseline for AL-CPL concepts due to its ability to handle out-of-vocabulary words with its sub-word representations[11]. These representations act as our word embedding baseline for CPL which will be compared to SMPNet Embeddings. The dimension of concepts' embeddings is $d_{FT} = 300$.

BERT Embeddings (Mean). To benchmark against RVGAE's method (Li et al., 2020) of generating embeddings using $BERT_{base}$ we replicated their approach for a fair comparison with our embeddings generated using SMPNet. As in RVGAE's method, we computed the element-wise average of BERT embeddings for each token of the concept's name, resulting in concept embeddings of dimension $d_{BERT_{Mean}} = 768$.

BERT Embeddings ([CLS]). For benchmarking against the TCPL method (Tang et al., 2023) which generates embeddings using BERT, we used the final hidden embedding of the special token [CLS] to represent concept pairs, resulting in **concept pair** embeddings of dimension $d_{BERT_{Mean}} = 768$. As outlined in the paper by Tang et al. (2023), sentences structured as '[CLS] concept_name_1 [SEP] concept_name_2 [SEP]' are fed into the BERT model. However, instead of using their specific pre-trained version of BERT, we employed the readily accessible $BERT_{base}$ model from Hugging Face, given that their model isn't publicly available. Moreover, compared to the $BERT_{base}$ model, their pre-trained BERT only improved their F-Score by 1%. Finally, since this method generates embeddings for concept pairs and not for individual concepts, it isn't compatible with our R-GCN-based architecture which takes as input single concept representations. Nonetheless, we feed both BERT embeddings ([CLS]) and SMPNet

[11] https://fasttext.cc/.

embeddings to an MLP classifier trained for CPL and compare their performance to assess which embedding representation is better suited for CPL.

Dimension Reduction. We apply PCA to reduce dimensions, retaining 95% of the variance. This results in dimensions of $d_{SMPNetPCA} = 197$ for SMPNet and $d_{FTPCA} = 161$ for FastText. These dimensions give us a good approximation of the optimal number of units for our GCN.

Model Baselines

We compared our proposed method for CPL with the following baseline models:

Zero Rule Algorithm. Predicts the class that appears most frequently in the training dataset. SOTA struggled to consistently beat the Zero Rule Algorithm in the cross-domain setting (Miaschi et al., 2019).

Bai et al. (2021). Although Bai et al. (2021) do not outperform the performance of Miaschi et al. (2019), it is interesting to compare their method to ours as they also use Transformers to represent concepts through textual data. We will only include the results of two of their models: their Automatic Features (AFs) model, which uses only BERT-generated embeddings, and their model using both AFs and Manually extracted Features from text (AFs + MFs). Their AFs model is comparable to the SMPNet embeddings + MLP Classifier as both of them rely solely on embeddings generated by Transformers. Their best-performing model in the in-domain setting is the AFs + MFs model. As they also adopt a 5-fold cross-validation to evaluate their models on AL-CPL in the in-domain setting, we can compare our results with theirs.

Miaschi et al. (2019). We also compare our model to models in Miaschi et al. (2019). Their M1 model utilizes pre-trained word2vec embeddings for binary label classification, M2 learns from manually extracted features from text, and M3 combines the inputs of M1 and M2. M1 is a good baseline to assess how other embedding methods perform on the CPL task. M3 achieves the best results so far on AL-CPL in the in-domain setting. Finally, M2 achieves the best results on AL-CPL in the cross-domain setting.

SMPNet embeddings + MLP Classifier. We draw inspiration from the Sentence-BERT model with a classification objective (Reimers & Gurevych, 2019) to create this baseline. We feed the triplet $(z_a, z_b, |z_a - z_b|)$ to an MLP classifier for CPL, where z_a and z_b are the SMPNet embeddings of concepts A and B. The weights of the MLP change during training, whereas

SMPNet concept embeddings are fixed. The optimal depth of the MLP is found through a hyperparameter search algorithm, as well as the dropout rate and the number of units per hidden layer. Its purpose is to gauge how much SMPNet embeddings are effective for CPL on their own.

PnPR-GCN without Domain Classification ($\boldsymbol{\mu = 0}$). This is our PnPR-GCN model that learns only the CPL task. We use it to measure the impact of concept domain classification in the in-domain and cross-domain settings.

4.3 Results

We report the results of our methods on the in-domain random split and the cross-domain CPL task in Table 4.1[12]. As described in Section 4.2.1, we use the 5-fold cross-validation strategy to generate all our results. We run our algorithms three times on the same, so we report the average result of the 15 runs for each value in Table 4.1.

Table 4.1: The average of the weighted F-Score results on the 5 AL-CPL test sets for in-domain and cross-domain CPL. Algorithm 1 is **not** used for fair comparison with previous results. The best results are in bold and marked with a **star***, while second best results are underlined.

In-domain						
	Architecture Type	**Data Mining**	**Geometry**	**Physics**	**Precalculus**	**Avg.**
Baselines	Zero Rule Algorithm	64.65	68.82	75.18	66.05	68.68
	SMPNet embeddings + MLP Classifier	**93.18***	94.13	**89.60***	**96.80***	**93.43***
	PnPR-GCN **without** Domain Classification ($\mu = 0$)	92.59	93.03	88.32	95.41	92.34
Bai et al. [3] (BERT-based)	AFs	83.3	89.4	76.7	89.9	84.83
	AFs + MFs	86.9	93.1	82.6	92.1	88.68
Miaschi et al. [22] (SOTA)	M1	88.81	92.43	83.43	92.48	89.30
	M3	89.66	**95.69***	88.54	94.95	92.21
Ours	PnPR-GCN	93.16	93.25	88.57	96.13	92.78
Cross-domain						
	Architecture Type	**Data Mining**	**Geometry**	**Physics**	**Precalculus**	**Avg.**
Baselines	Zero Rule Algorithm	64.65	68.82	75.18	66.05	68.68
	SMPNet embeddings + MLP Classifier	65.09	81.12	75.25	83.16	76.16
	PnPR-GCN **without** Domain Classification ($\mu = 0$)	67.38	82.97	79.02	85.24	78.65
Miaschi et al. [22] (SOTA)	M2	50.89	80.41	74.74	87.14	73.29
	M3	38.78	82.53	63.67	84.41	67.34
Ours	PnPR-GCN	**75.26***	**85.31***	**83.46***	**88.78***	**83.20***

For in-domain CPL, our SMPNet embeddings + MLP Classifier emerges as the top performer. Despite its simplicity, it surpasses previous methods that rely solely on textual data, such as the AFs and M1 models (refer to Table 4.1). It also outperforms SOTA methods that rely on large amounts of data, including the M3 and AFs + MFs models. This achievement underscores the effectiveness of our SMPNet embeddings + MLP Classifier and highlights the importance of leveraging contextual embeddings.

[12] In Miaschi et al. (2019), the weighted average F-Score is used as their metric. Indeed, for precalculus, 66.66% of the data is of class 0, and the Zero Rule baseline has an F-Score of 66.66%. If it was the macro averaged F-Score, the score of the baseline would be 0.50%. Thus, we report the weighted average F-Score obtained from our experiments.

An interesting observation arises from Table 4.1: Physics emerges as the most challenging domain for all approaches for in-domain CPL. Additionally, Physics exhibits the lowest percentage of prerequisite relations, with only 24.82% of concept pairs being prerequisites. This hints that class imbalance may render the problem harder to solve, despite the implementation of class re-balancing techniques across all the approaches in Table 4.1.

For cross-domain CPL, our GCN-based architecture surpasses previous approaches by an average F-Score of 9.91% in residual error as shown in Table 4.1. Specifically, in the domain of Data Mining, it achieves an increase of the average F-Score by 24.37%. These results are particularly encouraging as the performance of PnPR-GCN is not the best in the in-domain setting. Moreover, as hypothesized in Section 4.1.3, training on CPL and domain classification at the same time increases the average F-Score of PnPR-GCN by 4.55% on cross-domain CPL (see Table 4.1 for comparison of PnPR-GCN **without** domain classification ($\mu = 0$) and PnPR-GCN). Overall, we note the benefits of leveraging both graph representations and concept embeddings for cross-domain generalization.

Moreover, our findings highlight the challenges posed by Data Mining in the context of cross-domain CPL, as shown by the poor performance across all approaches. This difficulty stems from Data Mining having the lowest number of intersecting concepts with other domains (Table 3.2). On the other hand, Precalculus emerges as the easiest domain for CPL in the cross-domain setting, due to its highest number of intersecting concepts with other domains. These observations underscore the varying levels of difficulty across domains and emphasize the significance of concept intersections in addressing cross-domain concept prerequisite learning tasks.

4.3.1 Effect of Hyper Parameter Optimization on Performance

As mentioned in Section 4.1.3, we use grid search to find the optimal hyper parameters for our PnPR-GCN architecture. Each experiment is done on the 5 train/test splits, only the average F-Score on validation is reported.

We find the best hyper parameters for the in-domain setting for both random and graph split (see Table 4.2). The depth of the GCN embedding layer is 1 for both cases. As mentioned in Section 2.1, a GCN with l layers includes neighboring nodes up to a maximum distance equal to l in its aggregation to create node embeddings. Hence, because the average distance between nodes is close to 1 (refer to Table 3.6), $l = 1$ is enough to take into account all reachable nodes.

Table 4.2: Optimal hyperparameters for PnPR-GCN.

Split Type	**Average Distance Between Nodes**	**Number of GCN Embedding Layers**	**Size of GCN embeddings**	**MLP Depth**	**MLP Dropout Rate**
Random Split	1.121	1	675	2	0.8
Graph Split	1.0	1	764	2	0.7

Indeed, increasing the number of R-GCN embedding layers decreases the validation F-Score linearly (see left plot in Figure 4.6). Additionally, the uncertainty area, which is the area in blue around the line defined by the interval $(mean - standard\ deviation, mean + standard\ deviation)$, increases with the number of R-GCN layers, as overfitting becomes more probable. Moreover, the domain loss multiplier, $\mu \in \mathbb{R}$, impacts the validation F-Score (refer to Section 4.1.3 for the definition of the domain loss multiplier μ). For $\mu \in [0,1]$, the validation F-Score slowly increases to reach its maximum at $\mu = 1$. Then, for $\mu \in [1, 2]$, the validation F-Score increases and decreases unevenly. However, the maximum validation F-Score value remains reached at $\mu = 1$.

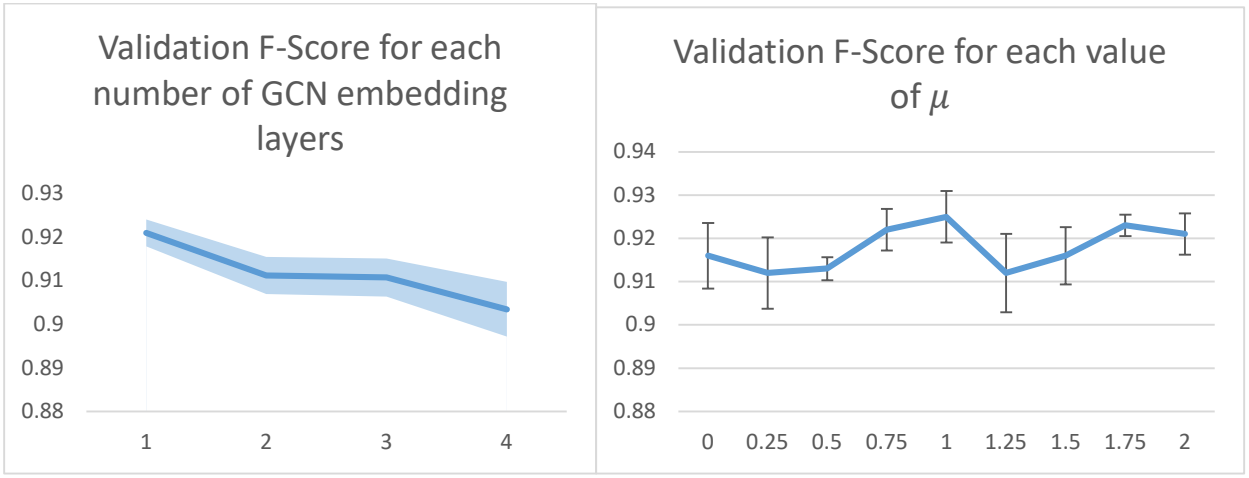


Figure 4.6: Validation F-Score of PnPR-GCN on the AL-CPL in-domain dataset per number of R-GCN embedding layers (left) and value of the domain loss multiplier μ (right).

In conclusion, the average distance between nodes is close to 1 for both split types, increasing the number of GCN embedding layers decreases the performance on CPL as expected. Additionally, domain classification improves CPL performance, and achieves optimal performance when both loss multipliers are equal to 1: $\mu = \lambda = 1$.

4.3.2 Ablation Study

We perform an ablation study to analyze the impact of different inputs on the performance of our PnPR-GCN architecture. The aim is to evaluate the contribution of graph representations and concept embeddings (see Table 4.3). We also measure the impact of solely using concept embeddings on CPL by training an MLP classifier with concept embeddings as an input (Table 4.4).

Table 4.3 presents the ablation study results of our PnPR-GCN architecture optimized to minimize both the CPL loss function and the domain loss function (see Section 4.1.3). As mentioned in Section 4.1, PnPR-GCN uses both the PnP concept graph (see section 3.1.2) and SMPNet embeddings (see section 4.1.2) as input. To measure the impact of both inputs on performance we conduct two types of ablation studies on PnPR-GCN. The ablation study without the SMPNet Embeddings in Table 4.3 replaces SMPNet embeddings with the three embedding baselines mentioned in Section 4.2.2: Random, BERT (Mean), and FastText without modifying the architecture of PnPR-GCN illustrated in Figure 4.2. For the ablation study without the PnP Graph in Table 4.3, we replace the PnP Graph with three other graphs: the Random Graph where the concepts are linked randomly, the prerequisite graph without transitivity, and the prerequisite graph with transitivity (refer to Section 3.1.2 for the definitions of last two graphs). The number of edges in the Random Graph matches the number of edges of the AL-CPL graph for each domain and edges are assigned randomly between nodes.

Table 4.3: **PnPR-GCN input ablation study.** Average over 5 folds generated by a random split for in-domain CPL of the test F-Score of GCN models using different types of concept embeddings (see Section 4.2.2) and graph representations (see Section 3.1.2).

Ablation Study Type	**Input Used**	**Avg. Test F-Score**
PnPR-GCN	**SMPNet Embeddings + PnP Concept Graph**	**92.78**
Without SMPNet Embeddings	Random	86.78
	BERT (Mean)	90.51
	FastText	91.27
Without PnP Graph	Random Graph	68.69
	Without Transitivity Graph	77.72
	With Transitivity Graph	84.62

On the first hand, Table 4.3 highlights the critical role that both text-based concept embeddings and graph representations play in the performance of GCNs on the CPL task. The randomization of concept embeddings and of graph representations significantly impairs the effectiveness of the GCN method. Moreover, we observe that the GCN architecture is more sensitive to the quality of its graph input compared to its node-level input. When exposed to a random graph, the GCN architecture performs similarly to the Zero Rule baseline.

On the other hand, both our choice of embedding methodology (SMPNet embeddings) and our choice of including non-prerequisite relations in the graph representation (PnP concept graph) proved to have yielded the best results according to Table 4.3. In particular, the inclusion of the non-prerequisite relations, which has never been done by previous approaches regarding CPL as a link prediction task (Li et al., 2020; Tang et al., 2023; Zhang et al., 2022), allowed us to increase the F-Score of PnPR-GCN by 8.12%.

In conclusion, the best setting to train PnPR-GCN is on the SMPNet embeddings and the PnP concept graph.

Table 4.4 presents the results of our experiment using only concept embeddings and an MLP Classifier. The objective of this ablation study is to evaluate the impact of concept embeddings alone, when they are not paired with a graph representation as in Table 4.3. We compare all the embeddings baselines mentioned in Section 4.2.2 (from Random Embeddings to FastText embeddings) with our SMPNet embeddings on equal footing: we feed the embedding pair of concept A and concept B, (A, B), to an MLP classifier which is trained to solve CPL in the in-domain setting as a binary classification task. Inspired by the training process of Sentence-BERT (Reimers & Gurevych, 2019), we add a third input to our MLP classifier. Therefore, we try $(A - B)$ as a third input (SMPNet Embeddings with $(A - B)$ as third input) and $|A - B|$ as third input (SMPNet Embeddings with $|A - B|$ as third input. Which is the baseline method described in Section 4.2.2).

Table 4.4: Average test F-Score of best MLP classifier on the AL-CPL dataset for the in-domain setting using different types of concept embeddings and input format.

Input Type Used	Avg. Test F-Score
Random Embeddings	68.69
BERT ([CLS]) Embeddings	86.48
BERT (Mean) Embeddings	88.53
FastText Embeddings	89.33
SMPNet Embeddings **without** |**A-B**| input	91.35
SMPNet Embeddings **with (A-B)** as third input	92.64
SMPNet Embeddings **with** |**A-B**| as third input (the baseline outlined in Section 4.2.2)	**93.43**

Results in Table 4.4 show that SMPNet embeddings are the most effective choice for CPL. The incorporation of the third input $|A - B|$ inspired by Sentence-BERT (Reimers & Gurevych, 2019) proved to be very effective. Notably, both our architecture and a standalone MLP classifier achieve their best results when using the SMPNet embeddings. We conclude that our method of generating concept embeddings using a sentence Transformer such as SMPNet surpasses other embedding methods used by previous approaches for CPL.

4.3.3 Concept Embeddings

Additionally, we propose a method to evaluate embeddings for our CPL task, drawing inspiration from recommender systems. Our evaluation process involves the following steps: First, we compute the cosine similarity matrix for all embeddings. Next, we rank the results for each concept in decreasing order of similarity. Finally, we calculate the evaluation metrics based on the definitions provided below:

$$\text{Precision@k} := \frac{\text{Number of Related Concepts in the top } \boldsymbol{k} \text{ results}}{\boldsymbol{k}},$$

$$\text{Recall@k} := \frac{\text{Number of Related Concepts in the top } \boldsymbol{k} \text{ results}}{\text{Number of Related Concepts}},$$

$$\text{MRR} := \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}. \tag{4.7}$$

Where for any given concept C, the term "related concepts" refers to either the prerequisites of the given concept C (ancestors) or the concepts for which C is a prerequisite (descendants). The query set, denoted as Q, contains all the related concepts to concept C.

Table 4.5: Precision at k, Recall at k, and Mean Reciprocal Rank (MRR) metrics for concept embeddings in the AL-CPL dataset.

Embedding Type	$\left\{\begin{matrix}\text{Precision}\\\text{Recall}\end{matrix}\right\}$@1	$\left\{\begin{matrix}\text{Precision}\\\text{Recall}\end{matrix}\right\}$@10	$\left\{\begin{matrix}\text{Precision}\\\text{Recall}\end{matrix}\right\}$@100	MRR (↑)
Random	0.062 0.006	0.066 0.078	0.064 0.737	0.036
BERT (Mean)	0.210 0.034	0.115 0.167	0.068 0.803	0.088
FastText	0.270 0.051	0.150 0.233	0.072 0.854	0.120
SMPNet	**0.542** **0.126**	**0.264** **0.468**	**0.078** **0.954**	**0.236**

In Table 4.5, we compare the three baseline methods for generating concept embeddings described in Section 4.2.2 with our method, which is SMPNet embeddings, with respect to the metrics defined in Equation 4.7.

The values presented in Table 4.5 are computed by averaging the metric values across all concepts and domains in the AL-CPL dataset. Once again, SMPNet embeddings prevail. Concepts with similar definitions are more likely to be linked by a prerequisite relation. SMPNet embeddings have the best performance in terms of CPL and the aforementioned metrics (see Equation 4.7). Indeed, higher values of MRR, Precision@k, and Recall@k approaching 1, indicate superior performance. Thus, given appropriate textual data, Transformers have the capability to generate semantically meaningful embeddings for concept prerequisite learning.

In Table 4.5, the scores for BERT Mean Generated embeddings on common recommender metrics for CPL are relatively low and underperform compared to the concept embeddings generated by FastText, which is a word embedding model. This confirms our criticism of the methodology behind the creation of BERT Mean embeddings (see Section 2.4.3). Additionally, this bad result of the BERT Mean Generated embeddings aligns with the findings of the paper from Reimers & Gurevych (2019).

In conclusion, the methodology for generating concept embeddings described in previous work underperforms compared to our methodology with respect to the metrics defined in Equation 4.7 (see Table 4.5) and the results of models using concept embeddings for CPL (see Table 4.3 and Table 4.4), which validates the claims we made in the Section 2.4.3 addressing the limitations of previous work in generating concept embeddings. However, even if our methodology for

generating concept embedding outperforms previous methods, it still has large room for improvement based on our results.

4.3.4 Results on the Graph Split AL-CPL

To test our hypothesis that due to the transitivity property the model evaluation in previous work is overly optimistic (Section 3.2.1), we conduct an experiment with different graph-based and transformer-based methods for solving CPL and measure the difference in model performance on the in-domain Random Split setting and the in-domain Graph Split setting (refer to Section 4.2.1).

Table 4.6 : In-domain CPL average weighted F-Score results on the test sets for Random and Graph split strategies.

			In-domain			
	Type	**Method**	**Random Split**	**Graph Split**	Δ_{Split} % (↓)	Δ_{SOTA} % (↓)
Literature	Bai. et al. (BERT-based) [3]	AFs	84.83	x	x	94.74%
		AFs + MFs	88.68	x	x	45.31%
	Miashi et al. (SOTA) [22]	M1	89.3	x	x	37.36%
		M3	92.21	x	x	0.00%
Our Results	Sentence Pair Classification	BERT[8]	90.93	89.54	15.33%	16.43%
		RoBERTa [21]	92.77	91.73	14.38%	-7.19%
		MPNet [29]	92.58	91.99	7.95%	-4.75%
		XLNet[35]	93.09	92.27	11.87%	-11.30%
	Our Methods	PnPR-GCN	92.78	92.33	**6.23%**	-7.32%
		SMPNet embeddings + MLP Classifier	**93.43**	**92.89**	8.22%	**-15.66%**

Table 4.6 summarizes the results on in-domain CPL using two different split methods on the AL-CPL dataset: the regular random split method used in previous work (see Section 3.2.1), and our Graph Split Algorithm (see Algorithm 1). We include additional Transformer-based methods to solve CPL on the in-domain Random Split and in-domain Graph Split settings. This is done in order to have a more significant sample size when measuring the difference in performance between methods trained on both types of splits, the Graph Split and the Random Split. In addition, they serve as a solid baseline for our approach of generating embeddings using Transformers. Therefore, we finetune BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), MPNet (Song et al., 2020), and XLNet (Yang et al., 2020) on CPL as if it was a sentence pair classification task (Devlin et al., 2019; Reimers & Gurevych, 2019).

To finetune Transformers on CPL as a sentence pair classification task, we fed pairs of concept-defining sentences such as '[CLS] concept_definition_1 [SEP] concept_definition_2 [SEP]' to

Transformers and used the final hidden embedding of the special token `[CLS]' to decide whether the first concept is a prerequisite to the second concept. All sentence pair classification methods are fine-tuned on AL-CPL using their base models.

Moreover, we include in Table 4.6 the average increase in residual error Δ_{Split} and Δ_{SOTA}:

$$\Delta_{\text{Split}} = \frac{100 - \text{FScore}_{\text{Graph}}}{100 - \text{FScore}_{\text{Random}}} - 1, \text{and } \Delta_{\text{SOTA}} = \frac{100 - \text{FScore}_{\text{Method}}}{100 - \text{FScore}_{\text{SOTA}}} - 1. \quad (4.8)$$

Where Δ_{Split} is the average increase of residual error between the average F-Score of a method on the in-domain Graph Split setting and the in-domain Random Split setting, and Δ_{SOTA} is the average increase of residual error between the average F-Score of a method on the in-domain Graph Split setting compared to the best SOTA results. In Table 4.6, smaller values of Δ_{Split} indicate that the model's performance degrades less when it is trained on splits generated by the graph split than on splits generated by the random split. For Δ_{SOTA}, negative values (highlighted in green) indicate that the model outperforms the best results of SOTA.

A first observation from Table 4.6 is that our Graph Split Algorithm described in Algorithm 1 renders the in-domain CPL task more challenging for all methods. It increases the residual error Δ_{Split} to 10.7% on average for all methods. As hypothesized in Section 3.2.1, preventing cases in the test set where the prerequisite inference is trivial due to transitivity makes the CPL task harder to solve. PnPR-GCN, SMPNet embeddings + MLP Classifier, and MPNet (Song et al., 2020) had the smallest increase Δ_{Split}. PnPR-GCN is the most robust at 6.23% while SMPNet embeddings + MLP Classifier and MPNet average around 8%. This is consistent with our previous findings in Section 4.3 that PnPR-GCN adapts better to harder training/testing settings such as the cross-domain setting.

Moreover, when fine-tuned for the CPL task using our concept-defining sentences, BERT (Devlin et al., 2019) outperforms Bai et al.'s (2021) AFs and AFs + MFs BERT-based methods. Using the Sentence-BERT (Reimers & Gurevych, 2019) framework with MPNet for CPL appears to be advantageous compared to their LSTM-based architecture in using contextual embeddings for CPL. Specifically, SMPNet embeddings + MLP achieves the best results with a $\Delta_{\text{SOTA}} = 15.66\%$, which is three times greater than what MPNet can achieve on its own (see Table 4.6). In summary, five of the six methods we trained on in-domain AL-CPL outperformed

the current benchmarks solely through the use of contextual embeddings and graph structures. PnPR-GCN has the lowest performance drop from the in-domain Graph Split setting compared to all methods.

CHAPTER 5 CONCLUSION

In this thesis, we present an R-GCN-based architecture for supervised Concept Prerequisite Learning on the AL-CPL dataset based on sentence Transformers and Graph Convolutional Networks (GCNs). Using pre-trained Sentence Transformers, we provide a methodology to generate vector representations for concepts from definitions that perform better than SOTA representations for CPL. Furthermore, in the context of the field of CPL, we introduce the first GCN-based architecture for CPL that jointly solves a link prediction and node classification task. Finally, we devise a new method to split the data and avoid trivial inferences of prerequisite relations in the test set. It addresses a methodological issue present in previous work.

5.1 Summary of the Contributions

Experiments show that our PnPR-GCN method performs better in the in-domain setting than the SOTA approach and previous methods. Additionally, our GCN architecture generalizes better in the cross-domain setting, as domain information on concepts is revealed to be crucial for cross-domain generalization. Furthermore, we show that our embedding generation method surpasses the embedding generation methods used in previous work for CPL. They outperform embeddings generated using the names of concepts and Transformers, showing that context matters for CPL. Finally, we test multiple architectures on the data split generated by our Graph Split Algorithm: on average, we observe a 10.7% decrease in performance across the board. Therefore, by excluding simple inferences by transitivity, our Graph Split Algorithm makes CPL harder to solve and more realistic.

5.2 Limitations

While our Graph Split Algorithm makes CPL a more realistic task, it has limitations that we plan to address in future work. A major limitation of the approach is that it neglects the asymmetry property inherent to prerequisite relations: if A is a prerequisite to B then B is not a prerequisite to A. As with transitive relations, a rule-based algorithm can easily make inferences based on this property, and our Graph Split Algorithm does not address this scenario.

Moreover, we solve CPL as a transductive link prediction task. Although it aligns with previous work, it makes GCNs less practical for the real-world problem of curriculum planning. Indeed, it is inconceivable to train an algorithm on all possible concepts in the educational system. New

concepts arise frequently, thus a more adaptive and robust approach to unseen nodes, such as inductive learning, would be more appropriate.

5.3 Future Work

In future work and regarding our Graph Neural Network approach, we would like to try a different approach and implement an inductive framework for prerequisite learning similar to GraphSAGE (Hamilton et al., 2018). An inductive GNN would be more suited for cross-domain generalization as it can generate node representations even for nodes unseen in the training set.

In addition, we would like to extend our experiments with CPL to datasets in languages other than English. The AL-CPL dataset itself is available in Italian and Chinese (Bai et al., 2021; Miaschi et al., 2019). This would help solidify our approach as adequate regardless of the language of the dataset.

Finally, future work plans to expand our approach to concept graphs from mined data to include unlabeled concepts in our input graph. We are exploring the possibility of using language models to identify prerequisites for unlabeled concepts as part of an unsupervised approach. The preliminary results indicate that this unsupervised approach is promising, as it achieved an F-score of 80% on the AL-CPL dataset without training the language model on the CPL task.

REFERENCES

Aleven, V. A. W. M. M., & Koedinger, K. R. (2002). An effective metacognitive strategy: Learning by doing and explaining with a computer-based Cognitive Tutor. *Cognitive Science*, *26*(2), 147–179. https://doi.org/10.1016/S0364-0213(02)00061-7

Ali, M., Berrendorf, M., Galkin, M., Thost, V., Ma, T., Tresp, V., & Lehmann, J. (2021). *Improving Inductive Link Prediction Using Hyper-Relational Facts* (arXiv:2107.04894). arXiv. https://doi.org/10.48550/arXiv.2107.04894

Bai, Y., Zhang, Y., Xiao, K., Lou, Y., & Sun, K. (2021). A BERT-Based Approach for Extracting Prerequisite Relations among Wikipedia Concepts. *Mathematical Problems in Engineering*, *2021*, 1–8. https://doi.org/10.1155/2021/3510402

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, *5*, 135–146. https://doi.org/10.1162/tacl_a_00051

Chaplot, D. S., Yang, Y., & Carbonell, J. (2016). *Data-driven Automated Induction of Prerequisite Structure Graphs*. 6.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., & Hsieh, C.-J. (2019). Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 257–266. https://doi.org/10.1145/3292500.3330925

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* (arXiv:1810.04805). arXiv. https://doi.org/10.48550/arXiv.1810.04805

Fenza, G., Gallo, M., Loia, V., Orciuoli, F., & Herrera-Viedma, E. (2021). Data set quality in Machine Learning: Consistency measure based on Group Decision Making. *Applied Soft Computing*, *106*, 107366. https://doi.org/10.1016/j.asoc.2021.107366

Graph Theory. (2008). https://link.springer.com/book/9781846289699

Hamilton, W. L., Ying, R., & Leskovec, J. (2018). *Inductive Representation Learning on Large Graphs* (arXiv:1706.02216). arXiv. https://doi.org/10.48550/arXiv.1706.02216

Hua, Y., Zhao, Z., Li, R., Chen, X., Liu, Z., & Zhang, H. (2019). Deep Learning with Long Short-Term Memory for Time Series Prediction. *IEEE Communications Magazine*, *57*(6), 114–119. https://doi.org/10.1109/MCOM.2019.1800155

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning* (Vol. 103). Springer. https://doi.org/10.1007/978-1-4614-7138-7

Jurafsky, D., & Martin, J. H. (2024). *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. https://web.stanford.edu/~jurafsky/slp3/ed3bookfeb3_2024.pdf

Kipf, T. N., & Welling, M. (2017). *Semi-Supervised Classification with Graph Convolutional Networks* (arXiv:1609.02907). arXiv. https://doi.org/10.48550/arXiv.1609.02907

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., & Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, *114*(13), 3521–3526. https://doi.org/10.1073/pnas.1611835114

Kuga, R., Kanezaki, A., Samejima, M., Sugano, Y., & Matsushita, Y. (2017). Multi-task Learning Using Multi-modal Encoder-Decoder Networks with Shared Skip Connections. *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, 403–411. https://doi.org/10.1109/ICCVW.2017.54

Li, I., Fabbri, A., Hingmire, S., & Radev, D. (2020). *R-VGAE: Relational-variational Graph Autoencoder for Unsupervised Prerequisite Chain Learning* (arXiv:2004.10610). arXiv. https://doi.org/10.48550/arXiv.2004.10610

Li, I., Fabbri, A. R., Tung, R. R., & Radev, D. R. (2018). *What Should I Learn First: Introducing LectureBank for NLP Education and Prerequisite Chain Learning* (arXiv:1811.12181). arXiv. https://doi.org/10.48550/arXiv.1811.12181

Liang, C., Wu, Z., Huang, W., & Giles, C. L. (2015). Measuring Prerequisite Relations Among Concepts. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1668–1674. https://doi.org/10.18653/v1/D15-1193

Liang, C., Ye, J., Wang, S., Pursel, B., & Giles, C. L. (2018). Investigating Active Learning for Concept Prerequisite Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1). https://ojs.aaai.org/index.php/AAAI/article/view/11396

Liang, C., Ye, J., Wu, Z., Pursel, B., & Giles, C. (2017). Recovering Concept Prerequisite Relations from University Course Dependencies. *Proceedings of the AAAI Conference on Artificial Intelligence*, *31*(1), Article 1. https://ojs.aaai.org/index.php/AAAI/article/view/10550

Liang, C., Ye, J., Zhao, H., Pursel, B., & Giles, C. L. (2018, January 19). *Active Learning of Strict Partial Orders: A Case Study on Concept Prerequisite Relations*. arXiv.Org. https://arxiv.org/abs/1801.06481v1

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). *RoBERTa: A Robustly Optimized BERT Pretraining Approach* (arXiv:1907.11692). arXiv. https://doi.org/10.48550/arXiv.1907.11692

Miaschi, A., Alzetta, C., Cardillo, F. A., & Dell'Orletta, F. (2019). Linguistically-Driven Strategy for Concept Prerequisites Learning on Italian. *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, 285–295. https://doi.org/10.18653/v1/W19-4430

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space* (arXiv:1301.3781). arXiv. https://doi.org/10.48550/arXiv.1301.3781

Pan, L., Li, C., Li, J., & Tang, J. (2017). Prerequisite Relation Learning for Concepts in MOOCs. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1447–1456. https://doi.org/10.18653/v1/P17-1133

Pennington, J., Socher, R., & Manning, C. (2014). GloVe: Global Vectors for Word Representation. In A. Moschitti, B. Pang, & W. Daelemans (Eds.), *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1532–1543). Association for Computational Linguistics. https://doi.org/10.3115/v1/D14-1162

Ploton, P., Mortier, F., Réjou-Méchain, M., Barbier, N., Picard, N., Rossi, V., Dormann, C., Cornu, G., Viennois, G., Bayol, N., Lyapustin, A., Gourlet-Fleury, S., & Pélissier, R. (2020). Spatial validation reveals poor predictive performance of large-scale ecological mapping models. *Nature Communications*, *11*(1), Article 1. https://doi.org/10.1038/s41467-020-18321-y

Rakesh Agrawal, and B. G. and E. P. (2015). *Data-Driven Synthesis of Study Plans* (TR-2015-003). Data Insights Laboratories.

Reimers, N., & Gurevych, I. (2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks* (arXiv:1908.10084). arXiv. https://doi.org/10.48550/arXiv.1908.10084

Rittle-Johnson, B., Star, J. R., & Durkin, K. (2009). The importance of prior knowledge when comparing examples: Influences on conceptual and procedural knowledge of equation solving. *Journal of Educational Psychology*, *101*(4), 836–852. https://doi.org/10.1037/a0016026

Roy, S., Madhyastha, M., Lawrence, S., & Rajan, V. (2019). Inferring Concept Prerequisite Relations from Online Educational Resources. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*(01), 9589–9594. https://doi.org/10.1609/aaai.v33i01.33019589

Scandura, J. M. (1966). Prior Learning, Presentation Order, and Prerequisite Practice in Problem Solving. *The Journal of Experimental Education*. https://www.tandfonline.com/doi/abs/10.1080/00220973.1966.11010947

Schlichtkrull, M., Kipf, T. N., Bloem, P., Berg, R. van den, Titov, I., & Welling, M. (2017). *Modeling Relational Data with Graph Convolutional Networks* (arXiv:1703.06103). arXiv. https://doi.org/10.48550/arXiv.1703.06103

Set Theory. (2003). Springer. https://doi.org/10.1007/3-540-44761-X

Sinha, S., Ohashi, H., & Nakamura, K. (2020). *Class-Wise Difficulty-Balanced Loss for Solving Class-Imbalance*. Proceedings of the Asian Conference on Computer Vision. https://openaccess.thecvf.com/content/ACCV2020/html/Sinha_Class-Wise_Difficulty-Balanced_Loss_for_Solving_Class-Imbalance_ACCV_2020_paper.html

Song, K., Tan, X., Qin, T., Lu, J., & Liu, T.-Y. (2020). *MPNet: Masked and Permuted Pre-training for Language Understanding* (arXiv:2004.09297). arXiv. https://doi.org/10.48550/arXiv.2004.09297

Tang, X., Liu, K., Xu, H., Xiao, W., & Tan, Z. (2023). Continual Pre-Training of Language Models for Concept Prerequisite Learning with Graph Neural Networks. *Mathematics*, *11*(12), Article 12. https://doi.org/10.3390/math11122780

Wang, S., Ororbia, A., Wu, Z., Williams, K., Liang, C., Pursel, B., & Giles, C. L. (2016). Using Prerequisites to Extract Concept Maps fromTextbooks. *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 317–326. https://doi.org/10.1145/2983323.2983725

Wu, Y., Zhao, S., & Li, W. (2020). Phrase2Vec: Phrase embedding based on parsing. *Information Sciences*, *517*, 100–127. https://doi.org/10.1016/j.ins.2019.12.031

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q. V. (2020). *XLNet: Generalized Autoregressive Pretraining for Language Understanding* (arXiv:1906.08237). arXiv. https://doi.org/10.48550/arXiv.1906.08237

Zhang, J., Lin, N., Zhang, X., Song, W., Yang, X., & Peng, Z. (2022). Learning Concept Prerequisite Relations from Educational Data via Multi-Head Attention Variational Graph

Auto-Encoders. *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, 1377–1385. https://doi.org/10.1145/3488560.3498434

Zhou, Y., Xiao, K., & Zhang, Y. (2020). An Ensemble Learning Approach for Extracting Concept Prerequisite Relations from Wikipedia. *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*, 642–647. https://doi.org/10.1109/MSN50589.2020.00107

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2020). *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks* (arXiv:1703.10593). arXiv. https://doi.org/10.48550/arXiv.1703.10593