

# Graph Neural Networks as the Copula Mundi between Logic and Machine Learning: a Roadmap

Andrea Agiollo, Giovanni Ciatto and Andrea Omicini

*Dipartimento di Informatica – Scienza e Ingegneria (DISI), ALMA MATER STUDIORUM—Università di Bologna, Italy*

## Abstract

Combining machine learning (ML) and computational logic (CL) is hard, mostly because of the inherently-different ways they use to represent knowledge. In fact, while ML relies on fixed-size numeric representations leveraging on vectors, matrices, or tensors of real numbers, CL relies on logic terms and clauses—which are unlimited in size and structure.

Graph neural networks (GNN) are a novelty in the ML world introduced for dealing with graph-structured data in a sub-symbolic way. In other words, GNN pave the way towards the application of ML to logic clauses and knowledge bases. However, there are several ways to encode logic knowledge into graphs: which is the best one heavily depends on the specific task at hand.

Accordingly, in this paper, we (i) elicit a number of problems from the field of CL that may benefit from many graph-related problems where GNN has been proved effective; (ii) exemplify the application of GNN to logic theories via an end-to-end toy example, to demonstrate the many intricacies hidden behind the technique; (iii) discuss the possible future directions of the application of GNN to CL in general, pointing out opportunities and open issues.

## Keywords

graph neural networks, machine learning, embedding, computational logic,

## 1. Introduction

Artificial Intelligence (AI) has gained significant importance in our ever evolving and technology-focused world. Rising popularity for this field can be attributed to the overwhelming success of sub-symbolic techniques like deep learning. However, along with AI increase in popularity, there exists public concern related to the relevant role that intelligent systems will bear in human society, in particular for the lack of understandability of AI systems.

Whenever intelligent systems play critical roles within human society, they should be made clearly understandable from a human perspective. This need has been taken under deep consideration by the XAI (eXplainable Artificial Intelligence [1]) research community. XAI approaches would seemingly require the integration between successful sub-symbolic techniques and symbolic frameworks in order to reach their main goals [2]. Among the symbolic approaches under scrutiny nowadays, logic-based techniques possibly represent the most straightforward path towards human understanding. The reason behind that is straightforward: symbols are far

---


WOA 2021: Workshop “From Objects to Agents”, September 1–3, 2021, Bologna, Italy

✉ andrea.agiollo@unibo.it (A. Agiollo); giovanni.ciatto@unibo.it (G. Ciatto); andrea.omicini@unibo.it (A. Omicini)

ORCID 0000-0003-0531-1978 (A. Agiollo); 0000-0002-1841-8996 (G. Ciatto); 0000-0002-6655-3869 (A. Omicini)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

closer to the way our conscious mind works than the vectors, tensors, and algebraic operations sub-symbolic AI is built upon. Along this line, it is essential for the future of AI to harmonise and integrate symbolic – and, in particular, logic-based – and sub-symbolic AI—and, in particular, neural networks.

The idea that symbolic and sub-symbolic AI are complementary under a number of dimensions is well understood [3, 4]. Few significant works have been proposed integrating neural networks with fuzzy logic [5, 6]. However, a general purpose solution to the problem is still lacking. Should we speculate on what is bringing inertia into this field of research, we would argue that the two approaches to AI are fundamentally dual w.r.t. the way they represent information—in short, formulæ vs. tensors. Indeed, a basic requirement of any integrated system involving both symbolic and sub-symbolic processing is the capability to either convert symbols into tensors, or vice versa—possibly both. This problem is hard to formalise in the general case.

In this work we focus on the simpler problem of enabling the sub-symbolic processing of logic knowledge-bases. In particular, we focus on the exploitation of neural networks (NN) as a means to complement computational logic (CL) when it comes to process symbolic data expressed in logic form. NN have indeed proven their strengths in many scenarios, ranging from computer vision to natural language processing, and, more recently, on knowledge graphs and graph-like data.

Accordingly, in this work we study the possible use of graphs as a bridge between CL – among the most prominent branches in symbolic AI – and neural networks—among the most flexible and successful approaches in ML. We consider graphs as the ideal bridge because of their versatility in representing virtually any sort of data structures, there including recursive ones—an aspect that is very common in logics as well as quite critical in ML.

Accordingly, we investigate the potential and the intricacies of blending CL and graph neural networks (GNN) [7]—a novel category of NN which is particularly well-suited to handle graph-like data. In particular, the aim of the work is to understand how and to what extent CL systems can benefit from GNN and graph-oriented ML in general. Along this line, the contributions brought about by this work are the following: *(i)* we categorise a number of problems from the field of CL which may take advantage from GNN, given the existence of a clear way to express the problem in terms of graphs; *(ii)* we discuss a number of graph-based tasks which can be suitably tackled via GNN, and discuss how the aforementioned problems can be mapped into such existing tasks; *(iii)* we present an end-to-end scenario where GNN are applied to a simple logic knowledge base to detect missing facts link prediction: the toy example is used to demonstrate the many intricacies hidden behind the application of GNN over logic data; *(iv)* finally, we provide a research roadmap that researchers interested in this field may follow for future works, eliciting opportunities arising from the integration between CL and GNN.

## 2. State of the Art

In this section we briefly introduce graph neural networks, along with the tasks these aim at solving and their working principle (Section 2.1). Then we give an overall background on computational logic, focussing in particular on the many ways to translate a knowledge base into a graph—in order to make it processable via GNN (Section 2.2).

## 2.1. Graph Neural Networks

In recent years, machine (ML) and deep learning (DL) techniques have disrupted the way complex data-driven tasks – ranging from image classification to speech recognition, natural language processing and many more – are tackled. However, most ML approaches can handle data having a *fixed* structure and size—most notably, vectors, matrices, or tensors of real numbers. This may be troublesome in some contexts, given the ever-increasing popularity of applications involving data which cannot be suitably represented by fixed-size, rigid structures. Among the most relevant applications in this category, we can find a number of *graph*-processing scenarios. To tackle this issue, research effort has focused on extending ML approaches to graph-structured data. Notably, *graph neural networks* (GNN) are a novel approach to let ordinary NN-based processing be applied to graphs.

GNN are mathematical models operating upon *directed* graphs, whose vertices (resp., arcs) are labelled with vectors (or matrices, or tensors) of real numbers, each one carrying further numeric information about the corresponding vertex (resp., arc). GNN output depends on the learning task to be performed, which commonly ranging in any of three wide classes of tasks: (i) the classification of similar graphs having different topology – i.e. *graph classification* – [8], (ii) the classification of vertices of unknown graphs – i.e. *nodes classification* – [9], or (iii) the identification of missing but statistically probable arcs—i.e. *link prediction* [10].

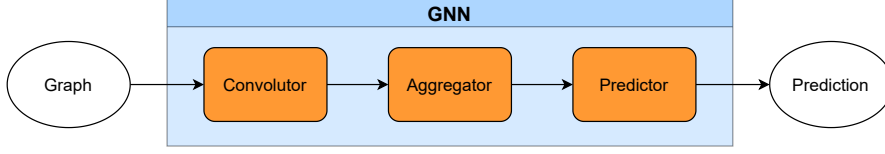
Graphs handled by GNN usually carry information in the form of vertices and arcs vectors/matrices/tensors. Consider for instance the graph representation of a chemical molecule: it is necessary to represent the sort of atomic element associated with each vertex. The same holds for the details of the chemical bonds among two any atoms of a molecule—which must be associated with the graph’s arcs.

Accordingly, we here consider vertices of a graph to be characterised by specific features called vertex attributes, represented as vectors of the form  $\mathbf{x}_v \in \mathbb{R}^d$ , where  $v$  enumerates the vertices of a graph, and  $d$  is the dimension of all vectors of all vertices. We also assume the existence of total ordering among the vertices in  $V$ , so we may refer to a vertex by its index  $i$ . We denote by  $\mathbf{X} \in \mathbb{R}^{n \times d}$  the matrix of all vertex attributes, attained concatenating each vector  $\mathbf{x}_v$  along a single dimension. There,  $n$  is the number of vertices in  $G$ . We also denote by  $N(v)$  the neighbourhood of a vertex  $v$ , here considered as the set of all vertices  $u$  directly linked to vertex  $v$  by an outgoing arc, i.e.  $N(v) = \{u \in V \mid (v, u) \in E\}$ .

Concerning arcs, we denote by  $a_{v,w} \equiv (v, w) \in A$  the arc connecting vertex  $v$  to vertex  $w$ . Similarly to vertices, we consider arcs in the graph as characterised by specific arc attributes, represented by vectors of the form  $\mathbf{a}_{v,w} \in \mathbb{R}^c$ , where  $c$  is the cardinality of the all vectors of all arcs. Finally, we denote by  $\mathbf{A} \in \mathbb{R}^{m \times c}$  matrix containing all arc attributes. There,  $m$  is the number of arcs in  $G$ .

Figure 1 depicts the general architecture of a GNN. It consists of a cascade of three functional blocks (each one composed by one or more layers of neurons) serving specific purposes:

**Convolutor.** The first block of the GNN is in charge of accepting the graph  $G$  as input and producing a new *convoluted* graph  $G'$  as output, having the same topology of  $G$ , where the vector associated with each vertex  $v$  has been replaced by another vector describing the relevance of each vertex w.r.t. the whole graph  $G$ . Convolutor block relies on convolution



**Figure 1:** Graph Neural Networks are composed as a cascade of simpler blocks.

operation, extensively exploited in DL to express relevance of local data w.r.t. to global data.

The application of convolution operation to non-Euclidean data – like graphs – is not straightforward. An equivalent notion of convolution over graphs has been proposed to compute the relevance of each vertex w.r.t. to its neighbours. Graph convolution is defined over a single vertex  $v$  and its neighbourhood  $N(v)$  and relies on three successive phases:

**propagation** – the information  $\mathbf{x}_{v'}$  belonging to each vertex  $v' \in N(v)$  is weighted by the information  $\mathbf{a}_{v,v'}$  belonging to the arc among  $v$  and  $v'$  and then propagated to vertex  $v$ ;

**aggregation** – the information propagated from each vertex  $v' \in N(v)$  to  $v$  is aggregated using a *parametric* aggregation function;

**transformation** – the aggregated information corresponding to vertex  $v$  is transformed into a new embedding vector and assigned back to vertex  $v$ , as its new state  $\mathbf{x}'_v$ .

The single convolution operation is applied in parallel to each vertex in  $G$ .

Inside the convulator block, the graph convolution procedure is repeated  $T$  times. The overall effect of step  $t$  is the production of a new graph  $G^t$  having the same vertices and arcs of  $G$ , where the  $i^{th}$  vertex at step  $t + 1$  carries a more convoluted information than the same vertex at step  $t$ . More formally, the relation tying each layer of the convulator block with its successor is captured by the following recursive equation:

$$\mathbf{x}_v^{t+1} = \Theta^t \left( \mathbf{x}_v^t, \uplus_{w \in N(v)} \Xi^t (\mathbf{x}_v^t, \mathbf{x}_w^t, \mathbf{a}_{v,w}) \right) \quad (1)$$

where functions  $\Xi^t$ ,  $\uplus$ , and  $\Theta^t$  represent the *propagation*, *aggregation*, and *transformation* phases respectively.

Function  $\Xi^t$ , in particular, propagates the information belonging to all neighbours  $w \in N(v)$  of vertex  $v$  through the arc that connects the two. This function must be differentiable and parametric – to be amenable of optimisation through the back-propagation algorithm [11] –, other than layer-specific and shared among all vertices of the graph.

Function  $\uplus$  aims at aggregating the information received by each vertex  $v$  from its neighbourhood. For this reason, it must be variadic and permutation invariant, other than being shared among all layers and all vertices of the graph.

Finally,  $\Theta^t$  is a differentiable, parametric, and layer-specific function, aimed at aggregating neighbourhood information to compute the vertex attributes for vertex  $v$  at step  $t$ .

**Aggregator.** The convoluted graph  $G'$  is passed to an aggregator block that produces a fixed-sized representation of the graph  $G'$ —called *embedding* of  $G$ ;

**Predictor.** The embedding produced by the aggregator block, being fixed in size, can be used as the input of an ordinary NN – namely, the predictor block – to solve ordinary ML tasks (e.g., classification or regression) on the original graph  $G$ .

## 2.2. Logic Theories as Graphs

Computational Logics (CL) essentially deals with logics as a means for computing [12]. Provided that knowledge can be expressed in terms of logic theories (a.k.a. knowledge bases, KB), made up of several logic clauses, CL endows software agents with automated reasoning capabilities, via many sorts of *inference* rules.

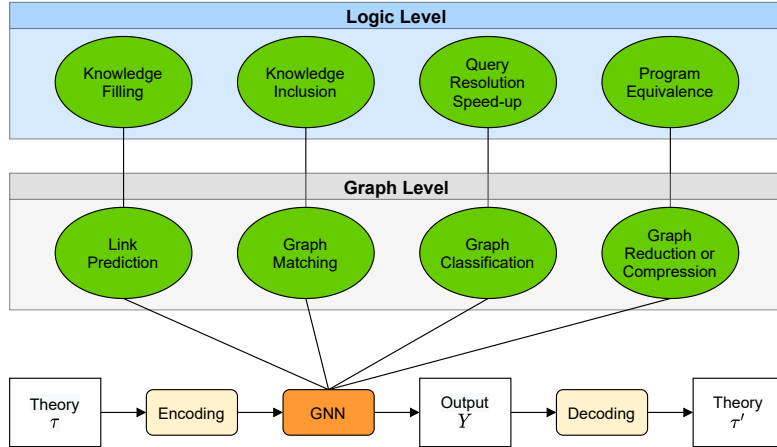
Knowledge bases can be encoded into graphs in several ways and to serve disparate purposes. Generally speaking, KB can be encoded into graphs by aggregating the graphs attained by encoding all clause therein contained. In all such cases, encoding schemas can act at either the *semantic* or at the *syntactic* level.

Encoding schemas operating at the *syntactic* level capture static relationships inferable from the mere syntax of clauses and KB. Abstract syntax trees (AST) are the simplest example of graphs which can be attained from KB. They consist of direct acyclic graphs where vertices are of as many sorts as the possible syntactical categories of which may occur in a KB – namely, theories, clauses, predicates, or terms –, whereas arcs simply describe container-contained relations among vertices. Dependency graphs are another kind of graph that may be attained from a KB. They consist of directed graphs where each vertex represents a predicate, and each arc represents a logic dependency among two predicates—meaning that the predicate corresponding to the destination vertex must be proven true before the predicate corresponding to the source vertex, in a resolution process.

Encoding schemas operating at the *semantic* level capture high level relationships that can be inferred from the actual meaning of a logic theory. Entity-Relationship (ER) graphs are the simplest kind of graph in this category. They aim at expressing via graphs the same information a ground KB expresses via formulæ. They consist of directed graphs where vertices may either represent entities (i.e. terms) or relationships (i.e. predicates) and arcs represent the participation of an entity into a relationship. Triplet graphs are another simple way of representing ground theories where all terms are constants and all predicates are either unary or binary. When this is the case, each constant is considered an entity, binary predicates are considered as relations among two entities, and unary predicates are considered as properties an entity may or may not have. Thus, a graph can be attained by defining a vertex for each different constant in a KB, and arc for each couple of constants involved in at least a binary predicate.

## 3. Processing Logic Knowledge via GNN

In this section we present a research roadmap eliciting the potential bridges among CL and GNN. We first identify four relevant tasks from CL where, we believe, sub-symbolic processing may have a role to play. We then discuss how all such tasks can be mapped into as many well-known graph manipulation tasks, for which GNN have already been exploited. Finally, we present a general framework for sub-symbolically processing logic information via GNN, and we elicit the many constraints a designer should satisfy when doing so.



**Figure 2:** Proposed roadmap along with its theoretical framework. Logic level and graph level can be mapped directly, in order for logical problems to benefit from sub-symbolical techniques – e.g. GNN – available at the graph level.

### 3.1. Logical Tasks

Manipulation of logic knowledge enables the resolution of complex queries via logical inference. There exist, however, relevant tasks which are hard to formalise or solve into the logic realm, because of either their numerical nature or algorithmic infeasibility. Here, in particular, we identify four relevant operations on knowledge bases for which, we argue, it is worth investigating sub-symbolic solutions.

The tasks considered – shown in the upper box of Figure 2 – are (i) knowledge filling, (ii) knowledge inclusion, (iii) program equivalence, and (iv) resolution speed-up.

**Knowledge Filling.** Entities and relations available in a logic theory may sometime lack some instances. For example, this may happen because the human operator handcrafting the theory was imprecise or when an agent’s knowledge is incomplete. When this is the case, we consider the knowledge base as *fragmented*.

To deal with such fragmented theories, it may be useful to identify missing relations between existing entities. This task may be tackled via statistical analysis of the theory under examination, which may lead to the identification of latent relations among entities.

A knowledge filling problem would be hard to handle symbolically, as logic reasoners commonly struggle in processing knowledge they do not have. While most solvers operate under a closed world assumption – letting them considers as false everything they do not explicitly know to be true –, even the ones operating under an open world assumption do not commonly include mechanisms to generate new knowledge out of thin air. In all such cases, the coherence and completeness of the knowledge base is usually considered as an a-priori requirement for logic computations to work properly. Conversely, in the sub-symbolic realm, semantic similarities among the entities and relations of a logic theory may be better captured, which may help reconstructing missing facts.

Consider for instance the case of a simple theory representing kinship relationships. The



lack of a single relation – say that “John and Mary are siblings” – may significantly hinder a solver’s ability to deduce kinships among entities of a family—e.g. “the sons of John and Mary are cousins”. The solution for this task is not straightforward, thus attracting our attention.

**Knowledge Inclusion.** Knowledge inclusion represents the task checking whether a given theory (usually smaller) is complementary w.r.t. another given theory (usually larger) or not. The same clauses could occur with slightly-different shapes—e.g. using different predicates/functor names or different positions arguments in the same predicates.

This task requires the ability to express equivalence or similarity among *groups* of clauses, which is not straightforward [13, 14]. Computing exact solutions to this problem may soon become infeasible as the dimensions of the involved theories increases. Conversely, in the sub-symbolic realm, the same problem may be modelled as a pattern-matching problem. This may pave the way towards the computation of *approximate* solutions to the knowledge inclusion problem in reasonable time.

As an example, consider multiple agents sharing partially-similar information. In this case, it would be desirable to identify agents common knowledge and ease their interaction. Suppose that agents information is expressed via two theories  $\tau_1$  and  $\tau_2$ , both representing family trees.  $\tau_1$  expresses 1<sup>st</sup> degree relatives only, while  $\tau_2$  includes also 2<sup>nd</sup> degree relatives.  $\tau_1$  may consider more/less/different family members w.r.t.  $\tau_2$ , and kinships may also be defined in different ways between the two theories. However,  $\tau_1$  is – logically speaking – a subset of  $\tau_2$ , and we need to detect this property.

**Program Equivalence.** Program equivalence represents the task of computing a simpler and equivalent theory  $\tau'$  starting from a theory  $\tau$ . This may imply removing redundancies and simplifying clauses. As for the knowledge inclusion task, program equivalence requires a procedure to compare sets of clauses, other than the capability of generating reduced equivalent variants of clauses. It is our opinion that both these procedures may be better expressed into sub-symbolical realm.

Considering again agents storing kinships information, it may be desirable to compress a single agent information to produce a new theory for a simpler agent. This new theory should ideally have fewer rules, while spanning the same family tree of the original theory. Such a requirement is difficult to satisfy, and would probably require notions of semantically-equivalent sets of kinships—e.g., the set of relations containing only *parent* spans the same family tree of the set of relations  $\{mother, father\}$ .

**Query Resolution Speed-Up.** Logic theories are commonly exploited by logic solvers to draw inferences, via some resolution procedure. The execution time of any query resolution vastly depends on the complexity of the algorithm(s) expressed by the logic theory. To this regard, a number of efficiency tweaks may affect the execution time in the average case. For instance, the solutions to most frequent queries may be cached, or smart strategies may be employed to affect the way the solver explores a solution space. However, caching costs space, whereas any *rigid* resolution strategy may result efficient on some sorts of queries, while still being slow on some others.

In all those cases, sub-symbolic sub-systems capable to learn from experience can bring about huge benefits. There, a sub-symbolic helper may be trained to predict the outcomes of most frequent queries, thus speeding up queries with constant space requirements. Furthermore, an online learning procedure may be injected into the solver, making it adapt the resolution strategy to the query at hand, on the basis of the experience accumulated via previous queries.

This task may be particularly relevant when considering real-time agents working with complex knowledge bases. Focusing again on kinships, when the number of family members is huge and relations between family members are complex – e.g., fourth grade cousins –, query resolution may suffer from delays hindering agents ability to make real-time decision and perform real-time tasks. Therefore, it may be interesting to use techniques that aim at speeding up the resolution of queries over such huge theories. Sub-symbolical approaches may ease this task, by compressing theory knowledge to simple and easy-to-handle embeddings.

### 3.2. Graphs as Bridges

Here we discuss the role of GNN in addressing the relevant logic tasks from Section 3.1. In particular, we show how all such tasks can be mapped onto known graph-related problems which can be addressed via GNN. In other words, we comment the upper part of Figure 2.

**Knowledge filling → Link prediction.** The knowledge filling task usually requires semantic knowledge to be taken into consideration. Therefore, to map the knowledge filling task to an equivalent problem over graphs we should consider preserving the semantic information of the theory. We can then assume to map entities of a theory to vertices of a graph. Rules and relations can then be represented as vectorised arcs existing between the graph vertices. Each position of an arc vector represents a specific relation, preserving the original semantic of the theory. In this scenario, the task of predicting possible missing relations or rules is mapped to the problem of identifying which arcs are missing from the graph.

**Knowledge Inclusion → Graph matching.** In the same way as for the knowledge filling task, knowledge inclusion requires semantic knowledge of the theory to be taken into account. Therefore, we require the mapping between logic and graphs to preserve the theory semantic. Moreover, knowledge inclusion requires a comparison between two or more theories: entities and relations from a theory should be compared to their counterparts of the other theory and matched upon need.

As done for knowledge filling, let us assume entities to be represented as vertices, and rules or relations as vectorised arcs. The mapping produces as many graphs as the theories available for the inclusion task. Therefore, from a graph perspective, knowledge inclusion is mapped to a graph matching problem. Indeed, the two or more graphs corresponding to their theory counterparts should be matched for some portion of them.

The matching between graphs is still an open research problem, as it is computationally very expensive, but is easier to tackle than rules and entities matching. This holds in particular whenever entities do not match exactly, or, rules share analogous semantics but are defined in different forms—e.g., parent and mother.



**Program equivalence**  $\rightarrow$  **Graph compression.** Given a specific theory, program equivalence aims at obtaining a simpler – smaller – theory that preserve the same expressiveness. Depending on the considered approach the mapping between logic and graph level may bear different requirements. In its simplest form program equivalence requires to simply remove unnecessary relations and rules of a theory to compress it. This approach does not require explicitly the semantic level to be considered while processing the theory. More interestingly, program equivalence may also require to map set of rules and relations to a single (or a smaller set of) rules(s). This increased complexity introduces the need for semantic to be taken into account and to be preserved in the mapping from logic to graphs. If we consider the same mapping of previous examples, program equivalence can be linked to the graph compression problem. Indeed, obtaining a smaller set of equivalent rules and entities can be done removing or merging together arcs and vertices of the graph theory counterpart.

**Query resolution speed-up**  $\rightarrow$  **Graph classification.** Query resolution speed-up aims at obtaining faster execution of given queries over a logic theory. It may be helpful for query resolution to maintain the semantic information embedded in the theory. Therefore, the mapping between logic and graphs may benefit from the preservation of semantic information, and generally speaking vastly depends on the requirements of the desired speed-up. Differently from previous tasks, for query resolution speed-up we consider obtaining graphs for queries to be solved—rather than a single graph for the whole theory. The graph representing a query is matched with the query resolution, considered as the graph label. Following this mapping, the query resolution speed-up is mapped to a graph classification problem, where the label of a graph should be predicted. Any approach can then be leveraged to classify graphs—i.e. obtain query solutions. This approach may not be significant for simple queries applied to small knowledge basis. However, it may result in great speed-up when complex knowledge bases and queries are considered. Indeed, GNN scalability over large graphs is mostly not an issue, resulting in quick graph classification.

### 3.3. The Framework Perspective

Here we summarise the general framework to process logic theories sub-symbolically, via GNN. In a nutshell, the whole framework consists in *transforming* the problem into the graph domain and let a GNN to do the job, then possibly *transform* back the problem into the logic domain. The same framework is depicted in the lower part of Figure 2.

Let us assume the overall goal of the whole processing, at a logic level, is to perform a task  $T$ —say, knowledge filling or inclusion. Let us also assume that an adequate mapping exists for  $T$  towards the graph level, such that  $T'$  is graph-related task corresponding to  $T$ , at the graph level. Under such assumptions, the framework involves the following steps:

1. a logic theory must be encoded into a graph using a suitable graph *encoding schema*;
2. a GNN must be designed and trained to perform  $T'$ , choosing
  - the functions  $\Xi^t$ ,  $\Theta$ , and  $\Theta^t$  for the GNN *convolutor* block,
  - the structures of the GNN *aggregator* and *predictor* blocks;

3. optionally, the output of the GNN shall be decoded back into a logic theory using a suitable graph *decoding schema*.

The emphasised words above represent choice points for the designer. While the possibilities are manifold, it is worth pointing out that each choice affects the others. For instance, while the encoding schema should be chosen by taking the nature of the logic clauses into account, the architecture of the predictor block, as well the choice of functions  $\oplus$  and  $\Theta^t$ , should be tailored on the task  $T'$ —and in particular on its nature under the learning perspective (e.g. whether  $T'$  is classification, regression, or clustering task).

Whether it is needed to perform step 3 (decoding) or not, is another source of constraints. There may be tasks – such query resolution speed-up, corresponding to graph classification – for which the outcome of  $T'$  is a Boolean datum, which needs not a decoding step. Conversely, other tasks may require the outcome of  $T'$  to be transformed back into the logic domain—cf. knowledge inclusion via link prediction. When this is the case, it is of paramount importance to choose an encoding schema which is *invertible*. This implies the encoding and decoding schemas are deeply entangled in the general case.

Summarising, symbolic processing may greatly benefit from the exploitation of sub-symbolic, GNN-based approaches. However, when this is the case, the overall data-processing framework must be carefully designed, as it involved may inter-dependent design choices.

## 4. Case Study

In this section we describe a case study that puts the theoretical framework introduced in Section 3 to test. We consider the knowledge filling task as the subject of this case study, as we believe it to be a nice introductory example to the world of logic manipulation using GNN. We proceed to set up our study case over a controlled environment, considering the knowledge basis representing kinship relations—i.e. family tree. We “mutilate” the knowledge base – meaning that we throw away a random part of the knowledge base – and use the remaining part to reconstruct the information removed by using the proposed framework.

### 4.1. Logic to Graph

As already mentioned, measuring the effectiveness of our framework requires to “mutilate” an otherwise exhaustive knowledge base. Theory mutilation can be then attained both at a logical level and at the graph level. In our experiments we mutilate the theory at graph level, to avoid multiple translations between the two levels. We now introduce the mapping function between logic level and graph level used in our experiment.

**Translation to Graph** Let  $\mathcal{C}$  be the set of all ground Horn clauses of the form  $h \leftarrow b_1 \wedge \dots \wedge b_m$  s.t. all  $b_i$  as well as  $h$  are predicates of arity non-greater than 2, and all arguments of all predicates are constant in  $\mathcal{H}$ . We consider  $\tau \in \mathcal{C}^*$  to be a ground theory containing  $N$  clauses and representing a family tree. We then define the *properties* of the theory  $\tau$  to be the set of all the unary predicates mentioned in all clauses of  $\tau$ . The set of properties is considered to be ordered through an index  $k$ , allowing to obtain a property calling it with the corresponding

```

sibling(X, Y) :- parent(Z, X), parent(Z, Y), X \= Y.
united(X, Y) :- parent(X, Z), parent(Y, Z), X \= Y.
grandparent(C,D) :- parent(C,E), parent(E,D).
aunt(X, Y) :- female(X), sibling(X, Z), parent(Z, Y).
uncle(X, Y) :- male(X), sibling(X, Z), parent(Z, Y).

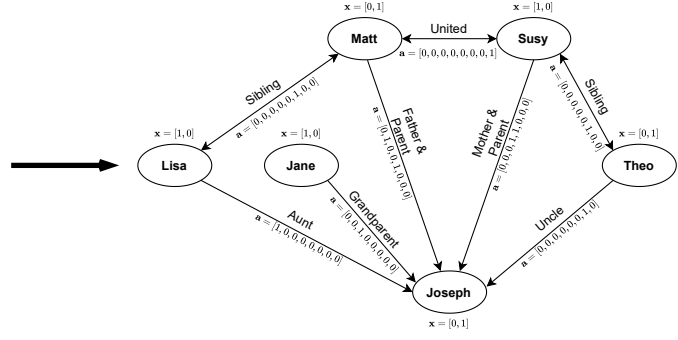
male(matt).
male(theo).
male(joseph).

female(susy).
female(lisa).
female(jane).

parent(matt, joseph).
parent(susy, joseph).
parent(jane, matt).

sibling(susy, theo).
sibling(theo, susy).
sibling(matt, lisa).
sibling(lisa, matt).

```



**Figure 3:** Example of the mapping function used in this case study. vertex attributes ( $\mathbf{x}_v$ ) represent unary predicates – i.e. properties – while arc attributes ( $\mathbf{a}_{v,w}$ ) represent binary predicates—i.e. relations.

index. We then define the *relations* of the theory  $\tau$  to be the set of all the binary predicates mentioned in all clauses of  $\tau$ . The set of relations is also considered to be ordered through an index  $l$ , allowing to obtain a relation calling it with the corresponding index.

To map the theory  $\tau$  to its graph counterpart  $G_{fill}$ , we first consider all the entities mentioned in all clauses of  $\tau$  and associate a vertex to each entity. Therefore, obtaining a graph  $G_{fill}$  with  $n$  vertices. Vertex features are then built as vectors  $\mathbf{x}_v \in \mathbb{R}^d$ , where  $d$  represents the size of the set of *properties* of  $\tau$ . Vector  $\mathbf{x}_v$  has value at position  $k \in \{1, \dots, d\}$  equal to 1 if property  $k$  holds true for entity  $v$  and 0 otherwise. The obtained vertex feature vector is thus a one-hot encoded vector representing the properties that characterise the entity. Similarly to vertices, arc features are built as vectors  $\mathbf{a}_{v,w} \in \mathbb{R}^c$ , where  $c$  represents the size of the set of *relations* of  $\tau$ . Vector  $\mathbf{a}_{v,w}$  has value at position  $l \in \{1, \dots, c\}$  equal to 1 if relation  $l$  between entities  $v$  and  $w$  holds true and 0 otherwise. The obtained arc feature vector is thus a one-hot encoded vector representing the relations satisfied for couples of vertices. Figure 3 exemplifies the mapping described above for a small family tree. In our experiment, predicates are the same of the figure, along with some added unary predicates – e.g. *has\_siblings*, *is\_parent*, *is\_grandparent*, etc. –, which are used to give more information concerning single entities.

Once  $G_{fill}$  is obtained, we proceed to mutilate the theory. Mutilation is attained removing some arcs – i.e. relations – between vertices of the graph. We call the graph obtained through this procedure  $G$  and the set of removed arcs  $A_{test}$ .

The proposed mapping allows constructing uniquely a graph from a grounded knowledge bases and is evidently bijective, as it is possible to reconstruct entities, properties and relations from  $\mathbf{x}_v$  and  $\mathbf{a}_{v,w}$ . However, it still presents some issues, as it requires groundisation of the knowledge bases and it can handle at most binary facts. The former can be considered a mild requirement as it is commonly considered for manipulation of symbolic knowledge. The latter instead has to be attributed to the nature of state-of-the-art GNN. Predicates of arity greater than 2 would require arcs linking more than two vertices at the time. Graphs having such links are called hypergraphs. These peculiar graphs are still an exception in the world of graph manipulation. There have been proposed very few solutions to handle these graphs [15], presenting strong drawbacks like the absence of arc features.

## 4.2. Graph Manipulation

Given the mutilated theory represented by the graph  $G$ , the task is to train a GNN model capable of mining the missing arcs. The GNN model is required to identify not only the existence of a missing arc between two vertices, but also to classify the arc into its class—i.e. which relation(s) the arc is representing. Due to graphs nature, link prediction can be tackled either considering  $G$  as a unique entity – i.e. *plain approach* – or as a pool of subgraphs—i.e. *subsampling approach*.

**Plain approach.** We consider the graph  $G$  as a whole and predict one solution for each couple of vertices. Indeed, each vertex can be involved in a relation with any other vertex of the graph. Here, we consider the set of arcs belonging to  $G$  as positive examples, called  $A$ . We then sample a set of negative examples  $\bar{A}$  as all the arcs that are not in  $A$ , nor in  $A_{test}$ . Given in input the graph  $G$ , a GNN model is then trained over  $A$  and  $\bar{A}$  to output two predictions:

- A binary matrix  $Y_e \in \mathbb{R}^{n \times n}$ . Where  $n$  is the number of entities in the graph. The value for position  $\{v, w\}$  is 1 if the GNN predicts that an arc should exist between vertex  $v$  and vertex  $w$  and 0 otherwise.
- A binary tensor  $Y_t \in \mathbb{R}^{n \times n \times c}$ . Where  $n$  is the number of entities in the graph and  $c$  is the number of available facts (kinship relations). The  $c$ -dimensional vector at position  $\{v, w\}$  corresponds to the one-hot encode of the arc type that the GNN predicts between vertices  $v$  and  $w$ .

The GNN model is composed of two graph convolutional layers that extract relevant information from  $G$  and produce a graph embedding  $G'$ . Given the need to predict a solution for each couple of vertices, we define as aggregation function the concatenation of vertices in the graph. The function is iterated over each couple of vertices  $v, w$  producing a vector  $\bar{x}_{v,w} = \mathbf{x}'_v \parallel \mathbf{x}'_w$  that represents the embedding for a possible arc between vertices  $v$  and  $w$ . Two parallel fully connected layers are then used as predictors to predict the existence – i.e.  $Y_e$  – of arc  $v, w$  and its type – i.e.  $Y_t$  – from  $\bar{x}_{v,w}$ . During training, cross-entropy loss  $\mathcal{L}_e$  is computed using  $Y_e$ , while binary cross-entropy over class types is  $\mathcal{L}_t$  is computed using  $Y_t$  [16]. The overall loss is then obtained through weighted summation of the two and used to optimize the GNN parameters.

$$\mathcal{L} = \mathcal{L}_e + \gamma \mathcal{L}_t \quad (2)$$

where  $\gamma$  is an hyperparameter balancing the importance of predicting arc existence or its type.

**Subsampling approach.** Similarly to [17], it is possible to consider the graph  $G$  as a pool of subgraphs each of which is used to predict if one arc exists. Here, one subgraph  $G_{sub}$  is obtained for each arc in  $A$ . For each arc  $a_{v,w}$ , the subgraph is obtained by keeping vertices  $v$  and  $w$ , as well as their neighbours  $N(v)$  and  $N(w)$ . The same sampling procedure is repeated for a set of negative examples  $\bar{A}$ . Therefore, following this approach we obtain a set  $\mathcal{G} = \{G_{sub}^1, \dots, G_{sub}^\nu\}$  of  $\nu$  graphs, each focused on an arc  $a_{v,w}$ .

A GNN model is then trained over  $A$  and  $\bar{A}$ , receiving in input a graph  $G_{sub}$  at the time, to output two predictions:

- A binary value  $Y_e \in \{0, 1\}$ . The value is 1 if the GNN predicts that the arc  $a_{v,w}$  should exist and 0 otherwise.
- A binary vector  $Y_t \in \mathcal{R}^d$ . Where  $c$  is the number of available facts. The vector corresponds to the one-hot encode of the arc type that the GNN predicts for arc  $a_{v,w}$ .

The GNN model is similar to the one of plain approach. It is composed of two graph convolutional layers that extract relevant information from  $G_{sub}$  and produce a graph embedding  $G'_{sub}$ . Given the need to predict a single solution for each graph, we define as aggregation function the global averaging pooling of vertices in the graph. Global average pooling produces in output a  $k$ -dimensional vector  $\bar{\mathbf{x}}$ , averaging vertex features of all vertices in the graph  $G'_{sub}$ ,  $\bar{\mathbf{x}} = \frac{1}{N} \sum_{v=1}^N \mathbf{x}'_v$ . Two parallel fully connected layers are then used as predictors to predict the existence – i.e.  $Y_e$  – of arc  $v, w$  and its type – i.e.  $Y_t$  – from  $\bar{\mathbf{x}}$ . GNN of both approaches are built using *PyTorch Geometric* [18] and *Deep Graph Library* [19]. Finally, loss computation remains the same of plain approach.

**Considerations.** It must be stressed that link prediction over graph is usually considered to be a binary prediction problem only. Indeed, state-of-the-art approaches focus only on the output  $Y_e$ . This is due to the nature of common link prediction applications – e.g. chemistry, social networking – where arcs between vertices belong mainly to one category only. As a consequence, the link prediction problem we face is more complex, as it represents the mixture of binary classification and multi-label classification—as there may exist multiple relations linking two entities. Tackling multi-label classification is not straightforward due to class overlapping and scalability issues [20].

It would be desirable to have relations semantically distant from each other to aid GNN in the multi-label classification task. Indeed, classes characterised by similar semantics are less separable and are probably subject to higher misclassification. Moreover, the scalability issue of multi-label classification hinders the performance of GNN when considering a high number of relation types. Although they could be overcome, these issues must be taken into account during the evaluation of the proposed experiment.

### 4.3. Results

During GNN training procedure we split either  $G$  (plain approach) or  $\mathcal{G}$  (subsampling approach) between training set and validation set. The former is used for backpropagation, while the latter is used to check GNN performance and save the best model. The best model obtained from training is then applied over  $A_{test}$  to test the final performance of the GNN.

We measure model performance over both predictions tasks—i.e. over  $Y_e$  and  $Y_t$ . We measure how the model behaves for the arc existence prediction task using the Area Under the Curve (AUC) [21, 22], Average Precision (AP) and accuracy metrics.

Evaluation of multi-label classification is not straightforward, as it introduces the notion of partially correct prediction—i.e. those predictions where a subset of the labels are identified correctly, but not all of them. To measure how the model behaves in the arc type prediction task we leverage the well known  $F_1$  score, the Exact Match Ratio (EMR), and the Per Example Accuracy (PEA). EMR ignores partially correct predictions, considering them as incorrect and

Approach	Edge Existence			Edge Class		
	AUC	AP	Accuracy	EMR	F <sub>1</sub>	PEA
Plain	0.786	0.786	0.786	0.143	0.148	0.768
Subsampling	0.908	0.892	0.906	0.691	0.386	0.911

**Table 1**  
Performance of the two different approaches over  $A_{test}$ .

computes the score as the ratio between correct predictions and total predictions. On the other hand, PEA computes the accuracy of the single sample  $i$  for each  $i$  in  $A_{test}$  and then average them together to obtain an overall accuracy metric.

Table 1 shows the performance of the two approaches on the link prediction task over  $A_{test}$ . Obtained results demonstrate the effectiveness of the proposed approach. The prediction of arc existence is successful with acceptable level of performance for the plain approach, while being highly successful when graph subsampling is applied. On the other hand, for the arc type prediction task we can notice the clear superiority of the subsampling approach.

Subsampling approach superiority can probably be attributed to the setup of the learning task. For the plain approach, the same input  $G$  is used to predict arcs belonging to training, validation and test, increasing the tendency of overfitting. On the other hand, subsampling approach allows to assign a different input  $G \in \mathcal{G}$  to each arc to predict. These inputs are different between training, validation and test, therefore allowing the model to train more easily, avoiding possible overfitting issues.

Given the number of possible arcs types (8) and their possible overlapping – e.g. *parent* and *father* –, the performance obtained by the subsampling approach are very satisfactory. The proposed model is capable to completely match an arc to its label – i.e. exact match over all 8 classes that define the arc – nearly 70% of the times, while the single arc types are predicted correctly more than 90% of the times. These results show the effectiveness of both the GNN model and the proposed theoretical framework of Section 3.

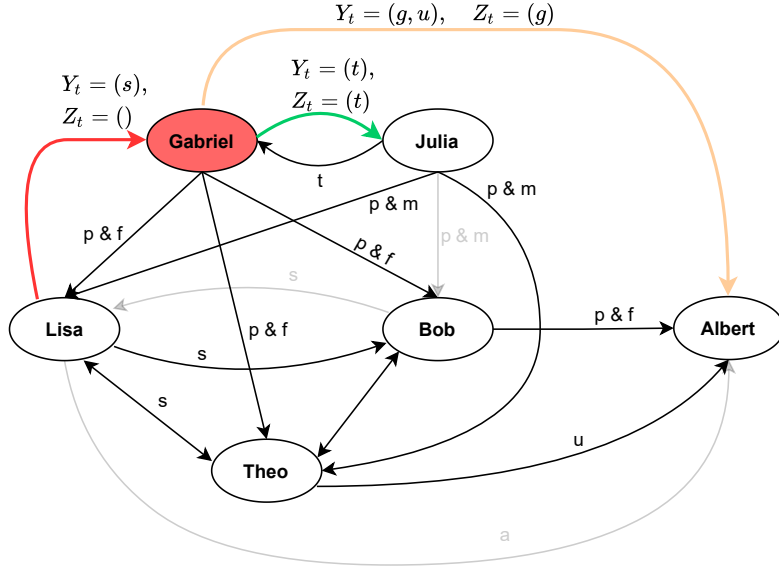
Figure 4 shows the three possible outcomes of arc predictions. An arc prediction may be completely correct—e.g. *Gabriel*  $\rightarrow$  *Julia*. There may also exist partially correct predictions—e.g. *Gabriel*  $\rightarrow$  *Albert*. Finally, *Lisa*  $\rightarrow$  *Gabriel* shows that there may exist arcs wrongly predicted.

Figure 4 also highlights a peculiar property of the model. The *Lisa*  $\rightarrow$  *Gabriel* prediction considered as wrong is actually a *daughter* relation. This prediction is considered as wrong since neither *daughter* nor *son* are defined in the original theory  $\tau$ . Indeed, to solve the knowledge filling task we rely only on the predicates already defined. However, the ability of the model of predicting this relation is sign of the GNN ability to understand the semantic of  $\tau$ . This GNN capability is encouraging, as it may help in tasks such as the discovery of new predicates.

**Role of overlapping classes.** As shown by Figure 3, the results obtained for Table 1 are influenced by class overlapping. Indeed, some arc types considered are semantically similar—e.g. *mother* and *parent*. Therefore, their vectorial representation produces label overlapping, which may negatively affect GNN performance.

To study the effect of semantically-similar classes on our approach, we consider a new theory





**Figure 4:** An example of three arc predictions involving entity *Gabriel*. Vertices represent entities and arcs represent relations, which can be aunt (a), father (f), grandparent (g), mother (m), parent (p), sibling (s), together (t), uncle (u). Black arcs and relations identify arcs in  $G$ , while gray arcs are arcs belonging to  $A_{test}$  and used for testing. Coloured arcs are the ones belonging to  $A_{test}$  and involving *Gabriel*. For these arcs  $Y_t$  and  $Z_t$  represent model prediction(s) and label(s) respectively. Green colour means the arc is predicted correctly, orange identifies partially correct arcs, while red pinpoint arcs wrongly predicted.

Approach	Edge Existence			Edge Class		
	AUC	AP	Accuracy	EMR	F <sub>1</sub>	PEA
<i>Plain</i>	0.857	0.857	0.857	0.286	0.214	0.738
<i>Subsampling</i>	0.949	0.961	0.938	0.670	0.340	0.935

**Table 2**

Performance of the two different approaches over  $A_{test}$  when overlapping arc types – e.g. *father* and *parent* – are not considered.

$\tau'$  identical to  $\tau$  – i.e., representing the same family tree –, but formulated without semantically-similar predicates. Practically speaking this requires to remove the notion of *father* and *mother* from  $\tau$  and redefine kinships using only the notion of *parent*.

We apply the same approaches of Section 4.2 to the new theory  $\tau'$ , and measure their performance. Table 2 shows the performance of the two approaches on the link prediction task over  $A_{test}$  when overlapping arc types are not considered. As expected, the superiority of the subsampling approach is unaffected. Instead, it is interesting to point out the effects of overlapping relations on the arc prediction task. The removal of overlapping arc types seems to affect positively *Edge Existence* prediction. Arc existence prediction compresses the knowledge of arc type to its mere existence. Therefore, overlapping relations might confuse the model in this scenario.

On the other hand, metrics related to the task of classifying arc type – i.e. *Edge Class* – seem to be untouched. This is a positive indication, as it hints at the fact that GNN can handle overlapping relations between entities. However, further study of this behaviour for the proposed model is required.

## 5. Conclusions

In this paper we propose a theoretical framework leveraging translation techniques between logic theories and graphs so as to tackle relevant logical problems. We identify the relevant logical problems and describe how they can benefit from the use of sub-symbolical approaches. Along with our framework, we identify possible mappings between logic and graphs, stressing their properties and defining how these should be identified by users. We then consider the filling of a fragmented theory as a study case and apply the proposed framework to this task. Obtained results show the goodness of our approach, introducing the possibility to leverage GNN to identify missing latent relations between entities of a theory. Finally, a brief study on GNN behaviour for overlapping classes in link prediction problem is presented, showing hints of GNN reliability.

Future works should focus on applying our framework for tackling relevant logical tasks in the CL world. Moreover, we consider relevant for future works to focus on some limitations of GNN, emerged from the work on our framework. Many logical tasks require mapping of predicates having arity greater than two to hypergraphs. Furthermore, some logical tasks require considering many relations, mapped to highly dimensional arc vectors. State-of-the-art GNN still suffer these requirements, especially when combined. We believe that future research in GNN should propose models capable of working on complex highly-dimensional hypergraphs. Finally, given the centrality role of arc relations in our framework, it would be desirable to have GNN models more focused on arc attributes relevance. Indeed, information is commonly updated only at vertices level, while logical tasks, and GNN performance more in general, would benefit from updating arc information as well. Few works so far have tackled this issue [23, 24], characterised by strong requirements and poor generalisability. Therefore, it exists research potential leveraging arc information more efficiently in GNN.

## Acknowledgments

This paper has been partially supported by (i) the H2020 project “StairwAI” (G.A. 101017142), and (ii) the CHIST-ERA IV project “EXPECTATION” (G.A. CHIST-ERA-19-XAI-005).

## References

- [1] A. B. Arrieta, N. D. Rodríguez, J. D. Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, F. Herrera, Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI, *Information Fusion* 58 (2020) 82–115. doi:10.1016/j.inffus.2019.12.012.

- [2] R. Calegari, G. Ciatto, A. Omicini, On the integration of symbolic and sub-symbolic techniques for XAI: A survey, *Intelligenza Artificiale* 14 (2020) 7–32. doi:10.3233/IA-190036.
- [3] F. J. Kurfess, Integrating symbol-oriented and sub-symbolic reasoning methods into hybrid systems, in: *From Synapses to Rules*, Springer, 2002, pp. 275–292.
- [4] M. L. Minsky, Logical versus analogical or symbolic versus connectionist or neat versus scruffy, *AI Magazine* 12 (1991) 34. doi:10.1609/aimag.v12i2.894.
- [5] L. Serafini, A. S. d’Avila Garcez, Logic tensor networks: Deep learning and logical reasoning from data and knowledge, in: T. R. Besold, L. C. Lamb, L. Serafini, W. Tabor (Eds.), *NESY 2016 – Neural-Symbolic Learning and Reasoning*, volume 1768 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016. URL: [http://ceur-ws.org/Vol-1768/NESY16\\_paper3.pdf](http://ceur-ws.org/Vol-1768/NESY16_paper3.pdf), proceedings of the 11th International Workshop on Neural-Symbolic Learning and Reasoning (NeSy’16) co-located with the Joint Multi-Conference on Human-Level Artificial Intelligence (HLAI 2016).
- [6] I. Donadello, L. Serafini, A. S. d’Avila Garcez, Logic tensor networks for semantic image interpretation, in: C. Sierra (Ed.), *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, Melbourne, Australia, August 19-25, 2017, ijcai.org, 2017, pp. 1596–1602. URL: <https://doi.org/10.24963/ijcai.2017/221>. doi:10.24963/ijcai.2017/221.
- [7] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P. S. Yu, A comprehensive survey on graph neural networks, *IEEE Transactions on Neural Networks Learning Systems* 32 (2021) 4–24. doi:10.1109/TNNLS.2020.2978386.
- [8] Z. Shui, G. Karypis, Heterogeneous molecular graph neural networks for predicting molecule properties, in: C. Plant, H. Wang, A. Cuzzocrea, C. Zaniolo, X. Wu (Eds.), *20th IEEE International Conference on Data Mining, ICDM 2020*, Sorrento, Italy, November 17-20, 2020, IEEE, 2020, pp. 492–500. doi:10.1109/ICDM50108.2020.00058.
- [9] H. Wang, J. Leskovec, Unifying graph convolutional neural networks and label propagation, *CoRR abs/2002.06755* (2020). arXiv:2002.06755.
- [10] W. Fan, Y. Ma, Q. Li, Y. He, Y. E. Zhao, J. Tang, D. Yin, Graph neural networks for social recommendation, in: L. Liu, R. W. White, A. Mantrach, F. Silvestri, J. J. McAuley, R. Baeza-Yates, L. Zia (Eds.), *The World Wide Web Conference, WWW 2019*, San Francisco, CA, USA, May 13-17, 2019, ACM, 2019, pp. 417–426. doi:10.1145/3308558.3313488.
- [11] R. Hecht-Nielsen, Theory of the backpropagation neural network, *Neural Networks* 1 (1988) 445–448. doi:10.1016/0893-6080(88)90469-8.
- [12] L. C. Paulson, Computational logic: its origins and applications, *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 474 (2018) 20170872. doi:10.1098/rspa.2017.0872.
- [13] T. Eiter, M. Fink, H. Tompits, S. Woltran, Simplifying logic programs under uniform and strong equivalence, in: V. Lifschitz, I. Niemelä (Eds.), *Logic Programming and Nonmonotonic Reasoning, 7th International Conference, LPNMR 2004*, Fort Lauderdale, FL, USA, January 6-8, 2004, *Proceedings*, volume 2923 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 87–99. doi:10.1007/978-3-540-24609-1\_10.
- [14] J. Ji, H. Wan, Z. Huo, Z. Yuan, Simplifying A logic program using its consequences, in: Q. Yang, M. J. Wooldridge (Eds.), *24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, AAAI Press, Buenos Aires, Argentina, 2015, pp. 3069–3075. URL:

<http://ijcai.org/Abstract/15/433>.

- [15] S. Bai, F. Zhang, P. H. S. Torr, Hypergraph convolution and hypergraph attention, *Pattern Recognition* 110 (2021) 107637. doi:10.1016/j.patcog.2020.107637.
- [16] J. E. Shore, R. W. Johnson, Properties of cross-entropy minimization, *IEEE Transactions on Information Theory* 27 (1981) 472–482. doi:10.1109/TIT.1981.1056373.
- [17] M. Zhang, Y. Chen, Link prediction based on graph neural networks, in: S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, 2018*, pp. 5171–5181. URL: <https://proceedings.neurips.cc/paper/2018/hash/53f0d7c537d99b3824f0f99d62ea2428-Abstract.html>.
- [18] M. Fey, J. E. Lenssen, Fast graph representation learning with pytorch geometric, *CoRR abs/1903.02428* (2019). arXiv:1903.02428.
- [19] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma, Z. Huang, Q. Guo, H. Zhang, H. Lin, J. Zhao, J. Li, A. J. Smola, Z. Zhang, Deep graph library: Towards efficient and scalable deep learning on graphs, *CoRR abs/1909.01315* (2019). arXiv:1909.01315.
- [20] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, W. Xu, CNN-RNN: A unified framework for multi-label image classification, in: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, IEEE Computer Society, 2016, pp. 2285–2294. doi:10.1109/CVPR.2016.251.
- [21] J. A. Hanley, B. J. McNeil, The meaning and use of the area under a receiver operating characteristic (ROC) curve, *Radiology* 143 (1982) 29–36.
- [22] A. Clauset, C. Moore, M. E. J. Newman, Hierarchical structure and the prediction of missing links in networks, *Nature* 453 (2008) 98–101.
- [23] L. Gong, Q. Cheng, Exploiting edge features for graph neural networks, in: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, Computer Vision Foundation / IEEE, 2019, pp. 9211–9219. doi:10.1109/CVPR.2019.00943.
- [24] X. Jiang, R. Zhu, P. Ji, S. Li, Co-embedding of nodes and edges with graph neural networks, *CoRR abs/2010.13242* (2020). arXiv:2010.13242.