Applied Soft Computing Journal 180 (2025) 113318

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Integrating a simplified formula graph representation into a graph neural network model for premise selection

Xingxing He [a], Zhongxu Zhao [a], Yongqi Lan [a], Yingfang Li [b,*], Li Zou [c], Jun Liu [d], Luis Martínez [e], Tianrui Li [f]

[a] School of Mathematics, Southwest Jiaotong University, Chengdu, 610031, China
[b] School of Computing and Artificial Intelligence, Southwestern University of Finance and Economics, Chengdu 611130, China
[c] School of Computer Science and Technology, Shandong Jianzhu University, Jinan, 250101, China
[d] School of Computing, Ulster University, Belfast BT15 1ED, Northern Ireland, UK
[e] Department of Computer Sciences, University of Jaén, Jaén 23071, Spain
[f] School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu 611756, China

ARTICLE INFO

Keywords:
Premise selection
Graph similarity
Graph representation
Graph neural network
SAGpool-Term-Walk

ABSTRACT

The search space for automatic theorem proving typically experiences exponential growth when attempting to prove a conclusion with numerous axioms. Premise selection presents a novel approach to tackle this challenge. However, one major obstacle lies in enhancing the presentation of logical formula graphs and graph neural network models in existing premise selection methods to preserve potential information from the logical formulas effectively. This study proposes a novel simplified graph representation of logical formulas by eliminating repeated quantifiers, along with a new term-walk graph neural network model incorporating an attention mechanism and attention pooling (ASTGNNS). This model aims to preserve syntax and semantic information of logical formulas, particularly regarding the order of symbols and the scope of quantifiers in logical formulas, thereby improving classification accuracy in premise selection problems. Specifically, we first transform first-order logical conjectures and premise formulas into simplified logical formula graphs by removing repeated quantifiers. Next, we introduce a method based on a common path kernel function to measure graph similarity and validate the interpretability of our simplified logical formula graphs method. Then, an attention mechanism is employed to assign weights to term-walk feature information of nodes for updating node feature representations; meanwhile, attention pooling is utilized for selecting nodes that significantly contribute towards generating the final formula graph vector. Finally, combining the premise graph vector and conjecture graph vector forms a binary classifier for classification purposes. Experimental results demonstrate that our proposed method achieves an accuracy rate of 88.77% on the MPTP dataset and 85.17% on the CNF dataset, outperforming the state-of-the-art premise selection method.

1. Introduction

Automatic theorem proving is a vital field in artificial intelligence, holding both theoretical significance and practical applications in expert systems [1], circuit design [2], compiler optimization [3], and software verification [4]. An automatic theorem prover (ATP) translates conjectures and premises into logical formulas to facilitate automated deduction. The ATP employs an iterative search strategy to generate proofs for new problems. Although it successfully addresses small-scale issues, large-scale libraries like the Mizar mathematical library [5] pose substantial challenges due to the extensive number of unrelated clauses and unproven propositions, resulting in computational demands that surpass current technological capabilities.

Premise selection offers a promising solution by identifying formulas that assist in proving a conjecture before they are entered into the prover. In recent years, effective premise selection methods have significantly improved ATP performance [6]. Early premise selection strategies primarily relied on hand-crafted heuristic methods based on symbolic comparative analysis [7]. These heuristic algorithms often faced limitations due to restricted feature design, which diminished their effectiveness in premise selection tasks. With advancements in computing power, machine learning techniques [8–11] such as convolutional neural networks [12], long short-term memory networks [13], and gated recurrent units [14] have emerged as effective alternatives for premise selection. These models excel at capturing deeper features

* Corresponding author.
E-mail address: liyf@swufe.edu.cn (Y. Li).

https://doi.org/10.1016/j.asoc.2025.113318
Received 10 August 2024; Received in revised form 12 March 2025; Accepted 11 May 2025
Available online 3 June 2025

of logical formulas. However, features derived from text sequences are limited in conveying information about logical symbols, while those based on tree structures mainly preserve the syntactic structure of logical formulas.

In contrast, graph-based representations can capture the syntactic structure and some semantic properties of logical formulas. Consequently, the combination of graph neural networks (GNNs) [15,16] and automatic theorem proving has emerged as a significant research area due to the natural representation of logical formulas as graphs. This approach allows comprehensive graph topological structure information integration to capture their features. Initially proposed by Wang et al. [17], a GNN model incorporating edge order information for premise selection was developed using a basic graph convolutional neural network (GCN) [18] for node embedding. Schlichtkrull et al. [19] expanded GCN by introducing a node relation weight matrix, resulting in a GNN model that preserves the relational information within logical formulas. Lin et al. [20] introduced an innovative representation method using contrast graphs for logical formula embeddings and incorporated an attention-based DenseNet convolutional network [21] to process the syntactic analytic graphs of formulas. Despite these advancements, existing research primarily focuses on GNNs' capabilities to encode graph information of logical formulas, with limited attention given to maintaining logical semantics within the graph representations. Consequently, the performance of GNN-based premise selection models is constrained by the encoding abilities of GNNs and the effectiveness of various methods used to represent logical formulas as graphs. Additionally, there has been an insufficient investigation into whether these graph representations are suitable for premise selection tasks, particularly regarding the theoretical justification for their applicability and interpretability. Moreover, most existing GNNs that rely solely on node and edge relationships capture only a limited set of properties of logical formulas, raising concerns about aggregating graph information in these networks to manage complex logical formula graphs. To address this issue, this paper proposes a simplified graph representation method that handles complex logical formulas and gives its interpretability.

Furthermore, the current research on graph representation has not addressed the consistency between logical formulas and their corresponding graphs. Intuitively, the graphs of logical formulas exhibit certain properties of both logical formulas and graphs. The kernel approach [22] offers a natural framework for measuring similarity. However, applying this approach to graph structure data remains challenging. Based on established graph similarity theory [23] and the characteristics of logic formula graphs, this paper redefines node labels and tailors graph similarity theory specifically for logical formula graphs. Consequently, we develop a method for measuring similarity in first-order logic formula graphs using common path kernel functions. We compute the similarity values between the enhanced and original logic formula graphs and compare various representations of logical formula graphs to the original, validating the feasibility of simplified first-order logic formula graphs at the level of graph similarity. Compared to existing graphs, our simplified first-order logic formula graphs eliminate redundant quantifiers, reducing data size while preserving essential quantifier properties. This streamlined representation demonstrates greater graph similarity with existing logic formula graphs, making it more suitable for automatic theorem-proving research and enhancing its interpretability.

To better incorporate logic formula graph information based on first-order logic formulas' internal and external attributes and highlight the significance of different node information, we propose an attention-based term-walk GNN that features an attention mechanism and attention pooling (ASTGNNS) and further investigate these properties before applying this methodology to premise selection tasks. The main contributions of this paper are as follows.

- We propose a simplified graph representation method for first-order logic formulas by eliminating repetitive quantifiers. This approach discusses the graph representation technique for reducing first-order logic formulas based on a large data scale and structure while preserving their logical properties.
- We propose a graph similarity measure utilizing a common path kernel function to evaluate the similarity between formula graphs. We demonstrate that the simplified graph representation method preserves more logical properties by analyzing the similarity between simplified and original formula graphs. This makes it particularly suitable for tasks related to automatic theorem proving. Consequently, we provide a theoretical justification for the interpretability of this simplified representation method.
- To address the simplified graph representation of first-order logical formulas achieved by eliminating redundant quantifiers, we propose an ASTGNNS model that comprehensively integrates term-walk feature information and node embedding information. This method generates a formula graph vector embedding that effectively captures the first-order logical formula's semantic and syntactic characteristics.

2. Related work

GNNs have emerged as a novel approach in machine learning, enabling effective extraction and exploration of features and patterns within graph-structured data. Furthermore, by leveraging their structural characteristics, first-order logic formulas can be represented as analytic trees. Consequently, the utilization of GNNs in premise selection surpasses traditional feature-based methods [24,25] and machine learning techniques [12–14], establishing them as mainstream methodologies in this domain.

Initially representing textual sequences primarily [12], the advent of GNNs has prompted researchers to investigate alternative graph representations for first-order logic formulas such as syntax trees [13, 21] and Directed Acyclic Graphs (DAG) [24]. DAGs are derived from the analytic tree structure of first-order logical formulas and encapsulate unique semantic and syntactic information. As a result, most premise selection models based on GNNs opt for DAG representation. To improve node information interaction in the formula syntax tree, Wang et al. [17] introduced the FormulaNet premise selection model based on higher-order logic formulas, which can transform the higher-order logic formula into a directed syntax tree with a root node. Subsequently, this directed syntax tree is converted into a DAG by merging identical subexpressions (subtrees) and applying variable renaming. A graph convolutional neural network that preserves edge order information is then used to obtain the graph feature embedding of the logical formula. Building on this work, Aditya Paliwal et al. [26] explored various graph representations of higher-order logic formulas. By analyzing logical formula graphs with or without leaf sharing, subexpression sharing, and variable renaming, they concluded that the logical formula graph representation, including subexpression sharing, performs better in premise selection tasks. At the same time, leveraging the satisfiability of logical formulas, several studies have examined graph feature representations for different representations of logical formulas. Xie et al. [27] used graph convolutional neural networks to encode decision deterministic decomposable negation normal form (d-DNNF) for logical formulas [28], introducing regularization techniques to ensure adherence to the two key properties of d-DNNF: certainty and decomposability. Given the absence of a unified structure for graph representations of logical formulas, Michael Rawson et al. [29] categorized existing graph representations into two main types: directed graph representations for propositional logic and argument order or variable binding representations for first-order logic, with the latter aligning with the graph representation [17].

Furthermore, Chaudhuri et al. introduced a clause-scoring neural network [30] that was trained using hindsight experience replay in

Table 1
Representative works of premise selection.

Approaches	Models	Structural characteristic	Symbols
Feature-based method	ENIGMA [25]	Tree representing	First-order logic
Mchine learning method	DeepMath [12]	Textual sequences	First-order logic
	Deep network model [13]	Syntax trees	First-order logic
	ENIGMA-NG [14]	Clause features	First-order logic
GNNs	FormulaNet [17]	DAG by merging identical subexpressions	Higher-order logic
	Subgraph pooling [26]	DAGs	First-order and higher-order logic
	Semisupervised GNN [27]	d-DNNF	First-order logic
	Directed graph networks [29]	Directed syntax graphs	Propositional logic first-order logic
	Clause-scoring neural network [30]	DAG	First-order logic
	NIAGRA [31]	Name-invariant formula representations	First-order logic
	Magnushammer [32]	Transformer architecture	Type systems
	Densely connected GCN [20]	Contrastive graph representation	First-order logic
	Attention-recurrent cross-GNN [37]	DAG	First-order logic

an incremental learning scenario. Fokoue et al. proposed an enhanced GNN for acquiring name-invariant formula representations [31], along with an efficient ensemble approach for automated theorem proving. Maciej et al. demonstrated that contrastive training with the transformer architecture [32] could yield superior retrieval of relevant premises. Bauer et al. devised a collection of benchmarking data sets for recommendation systems [33], aiding the formalization of mathematics with proof assistants. Lamont et al. presented a framework [34] to facilitate fair and streamlined comparisons of learning approaches in interactive theorem-proving settings. Yang et al. on the other hand, developed a large language model-based prover augmented with retrieval capabilities for selecting premises from an extensive math library [35]. For a more comprehensive survey on deep learning techniques applied to theorem proving, refer to Li's work [36].

To enhance existing premise selection methods, Michael Schlichtkrull et al. [19] introduced a node relationship weight matrix into GCNs, aiming to preserve the relationships between nodes in logical formulas as much as possible, thereby improving classification accuracy. Maxwell Crouse et al. [24] explored GNNs that retain both syntactic and semantic information of logical formulas, which proposed using GCNs and DAG long short-term memory networks (DAG LSTMs) to obtain node embeddings based on the reachability within directed graphs. A local attention mechanism was also introduced to facilitate continuous information exchange between nodes during training, updating the node embeddings accordingly. Lin et al. [20] developed a novel contrastive graph representation method for logical formula embedding, which employs a densely connected graph convolutional network with an attention mechanism to process the syntactic parse trees of formulas, generates contrastive formula graphs guided by different logical attributes, and applies both global-local and global-global contrasts to refine the formula embeddings. Liu et al. [37] introduced a premise selection method based on an attention-recurrent cross-GNN, which dynamically exchanges information among specific nodes during the aggregation phase and cyclically updates node embeddings while incorporating a gating mechanism based on node types to mitigate the impact of irrelevant information on the model's performance. Table 1 shows the representative works of premise selection.

Despite achieving state-of-the-art classification accuracy in premise selection, these GNN-based models primarily focus on the influence of various GNN architectures on premise selection and pay less attention to integrating the inherent properties of logical formulas, leading to limited retention of first-order logical formula information. Our proposed method builds upon an efficient GNN model to enhance the representation of logical formula graphs and the classification task of premise selection. We improve previous premise selection methods by refining the graph representation of logical formulas and optimizing the overall structure of the GNN through targeted modifications leveraging external and internal characteristics specific to logical formulas.

3. Simplified logical formula graph representation

This section presents a simplified representation of first-order logical formula graphs by eliminating repeated quantifiers (simplified DAGs). We specifically define graph similarity for logical formula graphs to clarify the relationship between simplified DAGs and DAGs.

Let an ordered binary set $G = (V, E)$ be a graph [38], where the nonempty set $V = \{v_1, v_2, \dots, v_n\}$ of G contains all the vertices and set $E = \{\langle v_i, v_j\rangle | v_i, v_j \in V\}$ contains all the vertices of G. $A = (a_{ij}) \in |V| \times |V|$ for the adjacency matrix of graph G [39] is defined as:

$$a_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E, \\ 0, & \text{otherwise,} \end{cases}$$

where $|V|$ is the number of vertices in G, a_{ij} is the number of paths from v_i to v_j that have length 1 and are not repeated.

For graph G, if the vertices v_i to v_j are reachable, then there is a path from v_i to v_j. If two paths of graphs G and G' have the same length, vertex labels, and edges, these paths are defined as the common paths of the two graphs.

Definition 1 ([23,40]). Let $x, x' \in X$, $X \in R(n)$, the non-linear function φ implements a mapping of input space X to feature space F, where $F \in R(m)$, $n \ll m$. The similarity between x and x' is calculated by kernel $k(x, x')$ as follows:

$$k(x, x') = \langle \varphi(x), \varphi(x') \rangle,$$

where $\langle , \rangle$ represents the inner product calculation.

Definition 2 ([23,40]). Let G and G' be graphs, and the common path kernel function $k(G, G')$ of G and G' is defined as follows:

$$k(G, G') = \mathbf{1} \cdot R \cdot \mathbf{1}' = \langle \varphi(G), \varphi(G') \rangle,$$

where $R = \sum_i R_i$, R_i is the common path matrix of length i of two graphs G and G', $\mathbf{1} = (1, 1, \dots)$ is a $1 \times n$ matrix with values 1.

3.1. Simplified first-order logical formula graph representation based on removing repeated quantifiers

By removing repeated quantifiers, a simplified first-order logical formula graph (referred to as simplified DAGs hereafter) can effectively reduce the size of the graph while accommodating more logical properties. Constructing simplified DAGs involves merging consecutive quantifiers based on the existing DAGs. Specifically, given a variable symbol set V, a function symbol set F, a predicate symbol set P, a logical conjunctive set C, a quantifier set Q, and an expression s representing a first-order logical formula (term, atom, or formula), Algorithm 1 recursively constructs simplified DAGs denoted by $G = (V, E)$ for the given first-order logical formula.

In Algorithm 1, the time and space complexities are strictly bounded by a linear order, specifically $O(m)$. The algorithm iterates over the m

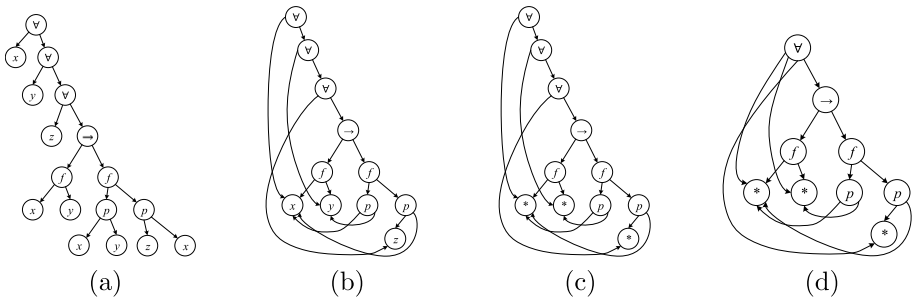

Fig. 1. The formula $\forall x\forall y\forall z(f(x,y) \to f(p(x,y),p(z,x)))$ is expressed as a simplified DAG; (a) Parse the first-order logical formula into a syntax tree; (b) Merge the same subexpression; (c) Replace all variables with *; (d) Merge the same quantifier nodes.

Algorithm 1 A simplified algorithm for constructing a directed acyclic graph from a formula in first-order logic

Require: A formula F in first-order logic, $A_1, \cdots, A_n$ are the logical symbols by their occurrences in F, $H(A)$ is the outermost logical symbol of A
Ensure: $G_F = (V_F, E_F)$ is the simplified directed acyclic graph and branch structure based on their connectives of F

Initiate $V_F = \{A_1\}, E_F = \emptyset$.
for $k \leftarrow 2$ to m **do**
  $V_F = V_F \cup \{A_k\}$; $E_F = E_F \cup \{< A_{k-1}, A_k >\}$
  if A_k is a constant symbol **then**
    if A_k is a term in a function symbol or predicate symbol $A_i (i \le k)$ **then**
      $E_F = E_F \cup \{< A_k, A_i >\}$
    end if
  else if A_k is a variable symbol **then**
    if A_k is a term in a function symbol or predicate symbol $A_i (i \le k)$ **then**
      $E_F = E_F \cup \{< A_k, A_i >\}$, $A_k = *$
    else if the occurrence of A_k is within the scope of the quantifier $A_j (j \le k)$ **then**
      $E_F = E_F \cup \{< A_k, A_j >\}$, $A_k = *$
    end if
  else if A_k is a function symbol or predicate symbol **then**
    if $A_k = f$ or $A_k = P$, and its terms are $A_{11}, \cdots, A_{1n}$ **then**
      $E_F = E_F \cup \bigcup_{i=1}^{n} \{< A_k, A_{1i} >\}$
    end if
  else if A_k is a quantifier **then**
    if $A_{k-1} = A_k$ **then**
      continue
    else
      $E_F = E_F \cup \{< A_k, x_1 >, < A_k, H(A_{k+1}) >\}$
    end if
  else if A_k is a logical connective **then**
    if $A_k = \neg$ **then**
      $E_F = E_F \cup \{< A_k, A_{k+1} >\}$
    else if $A_k = \odot$ and $A_k = H(A_{k-1}) \odot H(A_{k+1})$ **then**
      $E_F = E_F \cup \{< A_k, H(A_{k-1}) >, < A_k, H(A_{k+1}) >\}$
    end if
  end if
end for

logical symbols in the input formula for time complexity. The main loop runs for $m-1$ iterations, with each iteration's operations being completed in constant time $O(1)$, and no nested loops are involved. As for space complexity, the vertex set V_F holds m logical symbols. In contrast, the edge set E_F comprises $m-1$ edges derived from the initial graph structure, leading to an overall space complexity of $O(m)$.

The simplified DAGs proposed in this section satisfy the definition of a first-order logical formula graph, reducing its size while retaining some properties such as $\forall x\forall y f(x,y) = \forall y\forall x f(x,y)$ and $\exists x\exists y f(x,y) = \exists y\exists x f(x,y)$. Fig. 1 illustrates the process of representing a first-order logical formula $\forall x\forall y\forall z(f(x,y) \to f(p(x,y),p(z,x)))$ as a simplified DAG.

3.2. Graph similarity based on common path kernel function

The first-order logical formula can be transformed into corresponding DAGs on a one-to-one mapping. However, the simplified logical formula graph based on DAGs may inadvertently lose or misrepresent crucial information. Therefore, considering graph similarity, we aim to enhance the simplified representation's interpretability and applicability.

The kernel function, which relies on common paths, is closely associated with the graph's adjacency matrix [23], enabling it to capture structural information effectively. Nonetheless, this kernel function can only measure similarity between graphs with identical node labels. We propose an updated adjacency matrix to address this limitation and accommodate graphs with varying node labels.

Definition 3. Let the graphs $G(V,E)$ and $G'(V',E')$ be directed graphs, the updated adjacency matrix $\widetilde{A}$ and $\widetilde{A}'$ are defined as follows:

$$\widetilde{A} = A = a_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases}$$

$$\widetilde{A}' = a'_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E' \\ 0, & \text{otherwise} \end{cases}$$

where the adjacency matrix A and A' satisfy $\mathbf{1} \cdot A \cdot \mathbf{1}' \ge \mathbf{1} \cdot A' \cdot \mathbf{1}'$, and $\mathbf{1} = (1, \ldots, 1)$ is a $1 \times n$ matrix with values 1.

Based on a shared path, the kernel function computes the common matrix of two graphs using the adjacency matrix, thereby imposing constraints on the resulting common matrix. Consequently, the updated public path matrix is derived as follows.

Definition 4. Let $G(V,E)$ and $G'(V',E')$ be directed graphs, with A and A' representing their adjacency matrices respectively, the updated adjacency matrices $\widetilde{A}$ and $\widetilde{A}'$ accordingly. The common matrix R, which represents the Hadamard product of the updated adjacency matrices, is defined as follows:

$$R = \widetilde{A} \odot \widetilde{A}' = (a_{ij} \cdot a'_{ij}).$$

The nodes in the graph typically exhibit stronger interactions with nodes within relatively small neighborhood ranges [41]. Without loss of generality, the kernel function corresponding to a common path length of 1 is computed for illustration here. Therefore, we present the updated kernel function based on common path updates.

Definition 5. Let the graphs $G(V,E)$ and $G'(V',E')$ be directed graphs. The common path kernel function $k(G,G')$ of G and G' is defined as follows:

$$k(G,G') = \mathbf{1} \cdot R \cdot \mathbf{1}' = \langle \varphi(G), \varphi(G') \rangle,$$

where R is the common matrix for all paths of length 1 of G and G', $\mathbf{1} = (1, 1, \ldots)$ is a $1 \times n$ matrix with values 1, and $k(G,G')$ is the number of the common paths on the two graphs.

Selecting a fixed range for the kernel function is an appropriate approach to assess the similarity between any two sets of graphs in a dataset. Simultaneously, assigning specific weights to edge endpoints can effectively capture the significance and distinctions of different nodes and edges within the graph.

Definition 6. Let the graphs $G(V,E)$ and $G'(V',E')$ be directed graphs, A and A' their adjacency matrices, the weight matrix $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots)$. The similarity $R'(G,G')$ of G and G' is defined as follows:

$$R'(G,G') = \frac{\boldsymbol{\mu} \cdot k(G,G') \cdot \boldsymbol{\mu}'}{\boldsymbol{\mu} \cdot A \cdot \boldsymbol{\mu}'} = \frac{\boldsymbol{\mu} \cdot R \cdot \boldsymbol{\mu}'}{\boldsymbol{\mu} \cdot A \cdot \boldsymbol{\mu}'},$$

where R is the common matrix for all paths of length 1 of G and G', A and A' satisfy $\mathbf{1} \cdot A \cdot \mathbf{1}' \geq \mathbf{1} \cdot A' \cdot \mathbf{1}'$, and the value of each element in the weight matrix $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots)$ is not less than 0.

According to Definition 6, the kernel function $k(G,G') = \boldsymbol{\mu} \cdot R \cdot \boldsymbol{\mu}'$ is an effective method for calculating the inner product of corresponding eigenvectors. Consider two directed graphs, $G(V,E)$ and $G'(V',E')$, with adjacency matrices A and A' respectively, and a common matrix denoted as R. Consequently, the following conclusions hold.

Proposition 1. *For weighted graph similarity,* $R'(G,G')$, $0 \leq R'(G,G') \leq 1$.

Proof. From Definition 6, we have

$$R'(G,G') = \frac{\boldsymbol{\mu} \cdot k(G,G') \cdot \boldsymbol{\mu}'}{\boldsymbol{\mu} \cdot A \cdot \boldsymbol{\mu}'} = \frac{\boldsymbol{\mu} \cdot R \cdot \boldsymbol{\mu}'}{\boldsymbol{\mu} \cdot A \cdot \boldsymbol{\mu}'},$$

where the element values of matrix R and matrix A are both 0 or 1 and the value of each element in the weight matrix $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots)$ is not less than 0. Hence we have $\boldsymbol{\mu} \cdot R \cdot \boldsymbol{\mu}' \geq 0$, $\boldsymbol{\mu} \cdot A \cdot \boldsymbol{\mu}' > 0$ if and only if $R = 0$, $\boldsymbol{\mu} \cdot R \cdot \boldsymbol{\mu}' = 0$, $R = (a_{ij} \cdot a'_{ij}) \leq (a_{ij} \cdot a_{ij}) = (a_{ij}) = A$.

Therefore,

$$0 \leq R'(G,G') = \frac{\boldsymbol{\mu} \cdot R \cdot \boldsymbol{\mu}'}{\boldsymbol{\mu} \cdot A \cdot \boldsymbol{\mu}'} \leq \frac{\boldsymbol{\mu} \cdot A \cdot \boldsymbol{\mu}'}{\boldsymbol{\mu} \cdot A \cdot \boldsymbol{\mu}'} = 1.$$

Proposition 2. *For weighted graph similarity* $R'(G,G')$*, the following formula holds.*

(1) $R'(G,G) = 1$;
(2) $R'(G,G') = R'(G',G)$.

Proof. According to Definitions 5 and 6, the conclusions follow.

The value range of weighted graph similarity is observed to be [0,1] from Propositions 1 and 2, and its properties align with the general definition of similarity.

3.3. Weighted graph similarity of logical formulas

This section calculates the graph similarity of the first-order logical formula graph using the weighted graph similarity calculation method based on the updated common path kernel function. First, we establish the selection criteria for node label sets and define a weight matrix for the logical formula.

(1) Select the node label set of the logical formula graph

Let A and A' be the adjacency matrices of the formula graph $G(V,E)$ and $G'(V',E')$ respectively, then we have

$$V = (v_1, v_2, \ldots, v_n), V' = (v'_1, v'_2, \ldots, v'_m),$$

where the adjacency matrix A and A' satisfy $\mathbf{1} \cdot A \cdot \mathbf{1}' \geq \mathbf{1} \cdot A' \cdot \mathbf{1}'$.

The method for selecting the node label set, based on two logical formula graphs $G(V,E)$ and $G'(V',E')$, is provided due to the presence of repeated symbols among the nodes in the logical formula graph.

I. Two cases exist for the formula graph $G(V,E)$.

- If there are no repeated node symbols in V, then V remains unchanged.
- If there are repeated node symbols in V, let $i \in \mathbb{N}$ and v_i be the repeated node symbols, $\widetilde{V} = (\widetilde{v}_i)$ the extended label set of the repeated node concerning its child nodes. If there are still repeated node symbols in $\widetilde{V}$, expand the node label set until there are no same node labels. Then rewrite the V, that is,

  $$V = (v_1, \ldots, v_{i-1}, \widetilde{v}_i, v_{i+1}, \ldots, v_n).$$

II. For the formula graph $G'(V',E')$, two cases exist.

- If there are no repeated node symbols in V', then V' removes node symbols different from those in V, and we have

  $$(v'_1, v'_2, \ldots, v'_m) \to (\ldots, v_j, \ldots),$$

  where $v_j \in V$.
- If there are repeated node symbols in V', let $j \in Z$ and v'_j be the repeated node symbols, and $\widetilde{V}' = \widetilde{v}'_j$ the extended label set of the repeated node concerning its child nodes. If there are still repeated node symbols in $\widetilde{V}'$, continue to expand the node label set until there are no same node labels. Rewrite V' and remove the node symbol that is different from V, we have

  $$V' = (v'_1, \ldots, v'_{j-1}, v'_{j+1} \cdots, v'_m, \widetilde{v}'_j) \to (\ldots, v_j, \ldots, \widetilde{v}_i),$$

  where $v_j \in V$, $\widetilde{v}_i$ and $\widetilde{v}'_j$ are the same extended label set.

(2) Define rules of weight matrix oriented to logical formulas

Comparative analysis of multiple logical formula graphs shows that when a certain number of quantifiers are present, the corresponding part of the logical formula graph exhibits a highly similar structure. Conversely, if most logical formula graphs contain varying numbers of quantifiers, each graph will possess a similar structure.

To mitigate the impact of this aspect on classification tasks, we propose the following definition rule for the weight matrix: For a given formula graph $G(V,E)$, where $v_i (i \in \mathbb{N})$ represents either a quantifier symbol or an extended label containing quantifier symbols, the weight matrix can be defined as follows:

$$\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_i, \ldots, \mu_n) = (1, 1, \ldots, \frac{1}{4}, \ldots, 1),$$

where μ_i is the weight of the node v_i.

According to the established node label selection rules and weight matrix definition rules, the weighted graph similarity between two first-order logic formulas can be calculated using the specified formula. Assuming that $m = 4$ in the weight matrix $\boldsymbol{\mu}$, the calculation process for the weighted graph similarity of the two first-order logic formula graphs depicted in Fig. 2 is as follows: Figures (a) and (b) represent the logic formula graphs G and G', respectively, Figure (c) shows the adjacency matrix A of a graph G, Figure (d) illustrates the common matrix R derived from Definition 4 and Figure (e) presents the weight matrix and the weighted graph similarity obtained through Definition 6. It is important to note that each row of matrices A and R (from top to bottom) corresponds to the node labels $(\forall, x, \exists, y, \wedge, p, q)$.

3.4. Graph similarity between simplified DAGs and DAGs

To measure the overall similarity between simplified DAGs and DAGs, we compute the minimum similarity between them. The simplification process for DAGs involves removing identical and consecutive quantifiers from the logical formula graph. If DAGs cannot be simplified, the simplified formula graph based on deleting repeated quantifiers is identical to DAGs, resulting in a maximum similarity of 1. To evaluate the overall graph similarity between the simplified graph and DAGs, we calculate their minimum graph similarity. Given a logical formula graph G and its simplified version G' obtained by deleting

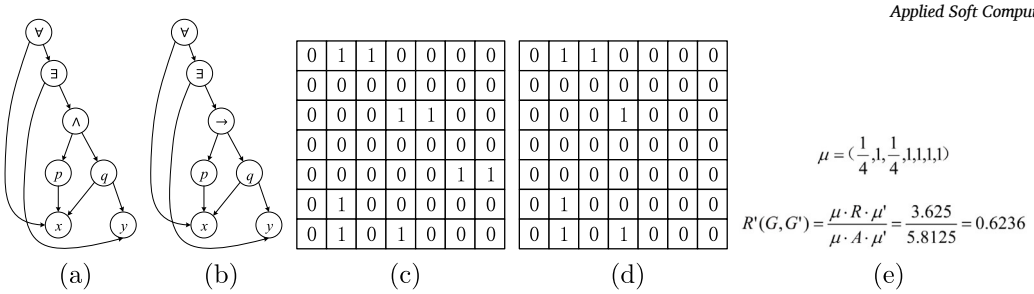

Fig. 2. An example of the first-order logic formula weighted graph similarity calculation process. (a) logical formula graph G; (b) logical formula graph G'; (c) the adjacency matrix of G; (d) common matrix R; (e) the weight matrix μ and the weighted graph similarity $R'(G, G)$.

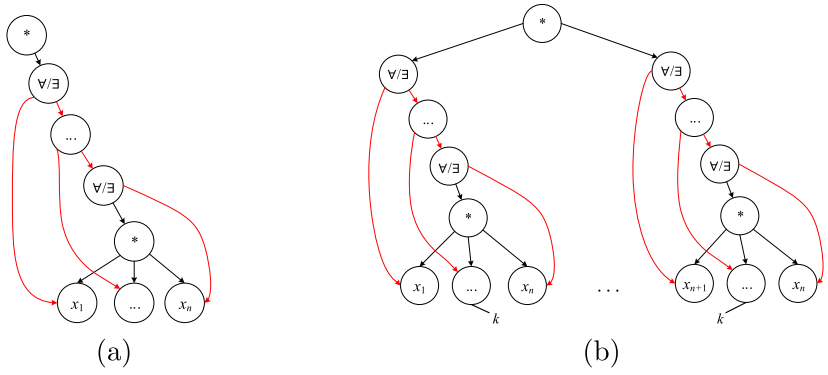

Fig. 3. Graph G with p quantifier nodes removed, $*$ denotes other logical symbols. Figures (a) and (b) show the logical formula graphs in different cases.

repeated quantifiers, let the adjacency matrix A of graph G satisfies $\boldsymbol{\mu} \cdot A \cdot \boldsymbol{\mu}' = n$, where the weight matrix $\boldsymbol{\mu} = (1, 1, \ldots, \frac{1}{m}, \ldots, 1)$. If DAGs can be simplified, the simplification process will remove edges corresponding to the quantifiers. Suppose p quantifier nodes are deleted. According to the structure of the logical formula, there are two cases, as illustrated in Fig. 3.

Case 1. The logical formula contains a sequence of consecutive repeated quantifiers with a total of $p+1$ repeated quantifiers.
If p quantifier nodes are deleted, graph G will be deleted $2(p+1)+1$ edges. Consequently, the weighted graph similarity based on the common path kernel function is

$$R'(G, G') = \frac{\boldsymbol{\mu} \cdot R \cdot \boldsymbol{\mu}'}{\boldsymbol{\mu} \cdot A \cdot \boldsymbol{\mu}'} = \frac{n - (\frac{1}{m} \cdot (p+1) + \frac{2}{m} + \frac{1}{m^2} \cdot p)}{n} = 1 - \frac{\frac{3+p}{m} + \frac{p}{m^2}}{n}.$$

If a fixed number of p quantifier nodes are deleted, the larger n is, the greater $R'(G, G')$ will be. We need to find the smallest n under the given p to calculate the minimum graph similarity. The minimum number of variables in a logical formula is $p+1$, and the minimum number of other logical symbols is 2. Therefore, the minimum number of edges in the logical formula graph is $3p+4$. Consequently, the minimum n can be calculated as

$$\min_n n = \frac{1}{m} \cdot (p+1) + \frac{2}{m} + \frac{1}{m^2} \cdot p + p + 1 = \frac{3+p}{m} + \frac{p}{m^2} + p + 1.$$

Substituting this n into the weighted graph similarity formula yields the weighted graph similarity when p quantifier nodes are deleted.

$$R'(G, G') = 1 - \frac{(3+p)m + p}{(3+p)m + p + m^2(p+1)} = 1 - \frac{1}{1 + \frac{m^2}{m+1+\frac{2m-1}{1+p}}}.$$

This equation shows that $R'(G, G')$ is a monotonically increasing function. Therefore, when p equals 1, $R'(G, G')$ attains its minimum value:

$$\min_{R'}(G, G') = 1 - \frac{4m+1}{4m+1+2m^2} = \frac{2m^2}{4m+1+2m^2}.$$

Case 2. The logical formula contains k groups of consecutive repeated quantifiers, with the total number of repeated quantifiers being $p+k$.
If p quantifier nodes are deleted, then the graph G will be deleted $2(p+k)+k$ edges. Consequently, the weighted graph similarity based on the common path kernel function is given by:

$$R'(G, G') = \frac{\boldsymbol{\mu} \cdot R \cdot \boldsymbol{\mu}'}{\boldsymbol{\mu} \cdot A \cdot \boldsymbol{\mu}'} = \frac{n - (\frac{1}{m} \cdot (p+k) + \frac{k}{m} \cdot 2 + \frac{1}{m^2} \cdot p)}{n} = 1 - \frac{\frac{3k+p}{m} + \frac{p}{m^2}}{n}.$$

Similarly, after deleting p quantifier nodes, the weighted graph similarity can be expressed as:

$$R'(G, G') = 1 - \frac{(3k+p) \cdot m + p}{(3k+p) \cdot m + p + m^2 \cdot (p+k)}.$$

This equation shows that $R'(G, G')$ is a monotonically increasing function. Therefore, when p equals k (noting that here k can be at most p), $R'(G, G')$ reaches its minimum value. Thus,

$$\min_{R'} R'(G, G') = \frac{2m^2}{4m+1+2m^2}.$$

Combining Cases 1 and 2, the minimum weighted graph similarity based on the common path kernel function can be calculated above. To mitigate the impact of repetitive structures, this paper sets $m = 4$ in the weight matrix $\boldsymbol{\mu}$. Hence, the minimum graph similarity is 0.653, indicating that this simplification method incurs a relatively low loss for DAGs.

3.5. Interpretation of simplified logical formula graph representation

Due to the diversity of logical formula symbols, various simplification methods exist. This section examines the similarity between

first-order logic formula graphs and DAGs based on different simplification techniques, culminating in a rationality analysis of simplified logic formula graphs derived from deleting repeated quantifiers. Specifically, it focuses on simplified first-order logic formula graphs derived from node deletions, including random node deletions and deletions of identical nodes.

Given an original logic formula graph G_0, we define three simplified formula graphs: G_1 based on random node deletion, G_2 based on deleting repeated quantifiers, and G_3 based on deleting identical nodes. Let the adjacency matrix A_0 of graph G_0 satisfy $\boldsymbol{\mu} \cdot A_0 \cdot \boldsymbol{\mu}' = n$, where the weight matrix $\boldsymbol{\mu}$ has $m = 4$. If p nodes are deleted, the following cases arise.

Case 1. Simplified logical formula graph based on random node deletion.
If p nodes are randomly deleted, particularly when p leaf nodes (all constant symbols) are removed, the DAG will have at least p fewer common edges. Therefore, the similarity measure $R'(G_0, G_1)$ can be expressed as:

$$\max R'(G_0, G_1) = \frac{\boldsymbol{\mu} \cdot A \cdot \boldsymbol{\mu}' - p}{\boldsymbol{\mu} \cdot A \cdot \boldsymbol{\mu}'} = \frac{n-p}{n}.$$

For the simplification method of deleting repeated quantifiers, the weighted graph similarity based on the common path kernel function is:

$$R'(G_0, G_2) = \frac{n - (\frac{5}{16} \cdot p + \frac{3}{4} \cdot k)}{n}.$$

- If $0 < k \leq \frac{11}{12}p$, then the minimum similarity $R'(G_0, G_2)$ is:

$$\min R'(G_0, G_2) = \frac{n - (\frac{5}{16} \cdot p + \frac{3}{4} \cdot \frac{11}{12} \cdot p)}{n} \geq \frac{n-p}{n} = \max R'(G_0, G_1).$$

- If $\frac{11}{12}p < k \leq p$, then the minimum similarity $R'(G_0, G_2)$ is:

$$\min R'(G_0, G_2) = \frac{n - (\frac{5}{16} \cdot p + \frac{3}{4} \cdot p)}{n} = \frac{n - (\frac{1}{16} \cdot p + p)}{n} < \frac{n-p}{n} = \max R'(G_0, G_1).$$

Case 2. Simplified logical formula graph based on deleting identical nodes.
Similar to Case 1, if p nodes are deleted, identical sub-expressions in DAGs will be merged, eliminating the possibility of deleting identical leaf nodes. Therefore, consider deleting consecutive identical nodes (excluding quantifiers), resulting in at least $p + 1$ fewer common edges in the DAGs. The maximum similarity is given by

$$\max R'(G_0, G_3) = \frac{n - (p+1)}{n}.$$

For the simplification method of removing duplicate quantifiers, the minimum weighted graph similarity based on the common path kernel function is

$$\min R'(G_0, G_2) = \frac{n - (\frac{1}{16} \cdot p + p)}{n} > \frac{n-(p+1)}{n} = \max R'(G_0, G_3),$$

where $p < 16$.

From Cases 1 and 2, we can conclude:

(1) If $0 < k \leq \frac{11}{12}p$, then the maximum graph similarity between the simplified graph based on random node deletion and DAGs is less than the minimum weighted similarity between the simplified graph based on deleting repeated quantifiers and DAGs. When p satisfies $p < 16$, the minimum graph similarity between the simplified graph based on deleting repeated quantifiers and DAGs is greater than the maximum graph similarity between the simplified graph based on deleting the same nodes and DAGs. This indicates that the simplified graph based on deleting repeated quantifiers has a higher graph similarity with DAGs in most cases.

(2) If $\frac{11}{12}p < k \leq p$, then the simplified graph based on random node deletion has a higher similarity. When p satisfies $p > 16$, the simplified graph based on deleting the same nodes has a higher similarity. However, the proportion of logical formula graphs that meet this form in large-scale problem libraries is small or non-existent.

Therefore, the simplified graph based on deleting repeated quantifiers is more similar to DAGs, shown as in Theorems 1 and 2.

Theorem 1. *Let G_0 be a logical formula graph, a simplified formula graph G_1 based on removing random nodes, and a simplified formula graph G_2 with removing repeated quantifier nodes, then $R'(G_0, G_1) < R'(G_0, G_2)$.*

Theorem 2. *Let G_0 be a logical formula graph, and a simplified formula graph G_2 with removing repeated quantifier nodes, a simplified formula graph G_3 based on removing the same nodes, then $R'(G_0, G_3) < R'(G_0, G_2)$.*

The graph representation of simplified first-order logic formulas, achieved by eliminating redundant quantifiers, shows greater graph similarity with DAGs. Within the first-order logic formulas, the graph similarity between distinct formulas is generally low due to inherent structural differences that often exceed simple graphical modifications. This observation indicates that the graph representation of simplified first-order logic formulas retains significant information from DAGs regarding graph similarity. Each graph representation derived from this simplification method can be effectively mapped to DAGs. Consequently, this approach preserves the essential characteristics of DAGs and is suitable for applications in automated theorem proving.

4. Model architecture

This section proposes a neural network model with attention mechanism and attention pooling based on a simplified logical formula graph (ASTGNNS) for premise selection. The overall architecture of a complete premise selection model based on ASTGNNS is shown in Fig. 4. The main part includes three branches, i.e., the graph representation of the first-order logical formula, the GNN model, and the binary classifier.

The present section proposes a neural network model incorporating attention mechanism and attention pooling, based on a simplified logical formula graph known as ASTGNNS, for premise selection. Fig. 4 illustrates the overall architecture of a comprehensive premise selection model built upon ASTGNNS. The primary components consist of three branches: the graph representation of first-order logical formulas, the GNN model, and the binary classifier.

- **Graph representation module**: The model receives a sequence of first-order logical formulas and constructs a simplified graph representation by eliminating redundant quantifiers.
- **GNN module**: This module utilizes term-walk mode to aggregate node information and incorporates an attention mechanism for assigning varying weights to the term-walk features of nodes. It further aggregates node information based on the weighted term-walk feature data. Attention pooling, average pooling, and maximum pooling techniques generate the final vector representing the first-order logical formula graph.
- **Binary Classifier module**: This module concatenates the premise graph vector and conjecture graph vector into Multi-Layer Perceptrons (MLPs) with classification capability. It outputs a predicted classification score between 0 and 1, which is used for classifying the premise.

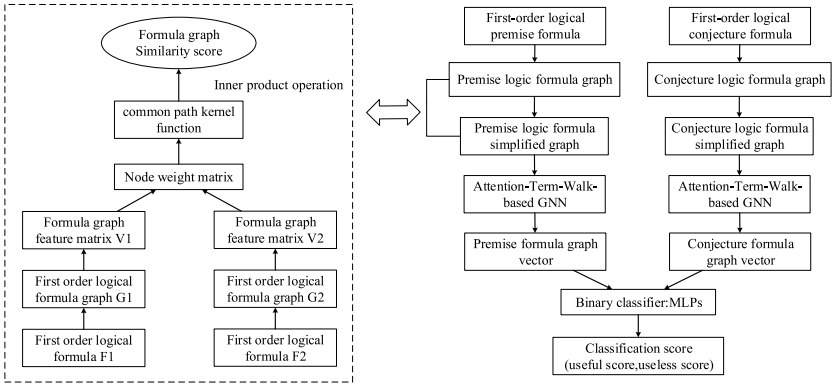

Fig. 4. The model architecture of ASTGNNS.

4.1. Term-walk graph neural network with attention mechanism and attention pooling

Our proposed ASTGNNS model simplifies the representation of first-order logical formulas by eliminating repeated quantifiers applied to a GNN. The GNN module of our model draws inspiration from the Message-Passing Neural Network (MPNN). Additionally, our GNN model incorporates the existing term-walking feature based on the analytical tree of logical formulas and generates the final embedded formula graph vector by introducing an attention mechanism and attention pooling.

ASTGNNS consists of four stages: initialization of graph node vectors, aggregation of graph node information, transfer of graph node information, and aggregation of overall graph information (graph pooling). Through iterative updates to the node embedding information and application of graph aggregation operations, ASTGNNS obtains the ultimate embedding vector for first-order logical formula graphs.

4.2. Term-walk feature

The term-walk feature can be applied to various tasks of automatic theorem proving, and the term-walk feature based on the analytic tree of logical formulas can be easily extended to the term-walk feature based on simplified DAGs. More formally, let $G = (V, E)$ represent a DAG. The term-walk feature in G encompasses all directed paths of length 3 within the graph G. Additionally, all term-walk features in G consist of triplets $(u, v, w) \in V \times V \times V$, where $v \in V$ represents any node in the graph, $u \in V$ is the parent node of v, and $w \in V$ is the child node of v. The logical formula $\forall x(f(x) \wedge p(x))$ serves as an item within a characteristic collection containing $\{(\forall, \wedge, f), (\forall, \wedge, p), (\wedge, f, x), (\wedge, p, x)\}$.

4.3. Graph node vector initialization

The ASTGNNS model begins by assigning an initial embedding x_v to each node $v \in V$ in the input graph $G = (V, E)$. Subsequently, it implements a message passing process through the node state vector $h_v^{(k)}$, which is generated by iterating for K rounds ($k \in \{1, \ldots, K\}$). During the initialization stage of the graph node vector, the model maps the initial eigenvector x_v of any given node v to a fixed-size initial state vector h_v^0 using an initial embedding function denoted as F_V.

$$h_v^0 = F_V(x_v),$$

where $h_v^0 \in R^{d_{h_v}}$ is the output dimension of the node embedding vector, and F_V is a lookup table that stores a fixed dictionary and size embedding.

4.4. Graph node information aggregation

The node information aggregation module of ASTGNNS is based on the term-walk feature information of nodes, and an attention mechanism is incorporated to generate weighted node term-walk feature information as shown in Fig. 5. The node information aggregation module collects node information based on their respective term-walk feature sets. Specifically, for each node V in the input graph $G = (V, E), T_u(v), T_m(v), T_l(v)$ represent the term-walk feature sets of nodes v located in the upper, middle, and lower parts respectively.

$$T_u(v) = \{(v, u, w) | (v, u), (u, w) \in E\},$$
$$T_m(v) = \{(u, v, w) | (u, v), (v, w) \in E\},$$
$$T_l(v) = \{(u, w, v) | (u, w), (w, v) \in E\}.$$

The TW-GNN model aggregates node information by calculating the average value of node term-walk feature information while disregarding the varying importance of different term-walk features. To address this issue, ASTGNNS introduces an attention mechanism to compute the term-walk feature scores $s_u^{(k-1)}$, $s_v^{(k-1)}$, and $s_w^{(k)}$ for each node, and utilizes the softmax function to normalize these scores into attention weights α_u, α_v, and α_w.

$$s_u^{(k)} = \sigma\left(\left[h_v^{(k-1)}; h_u^{(k-1)}; h_w^{(k-1)}\right]\right), \alpha_u = \mathrm{softmax}\left(s_u^{(k)}\right),$$
$$s_v^{(k)} = \sigma\left(\left[h_u^{(k-1)}; h_v^{(k-1)}; h_w^{(k-1)}\right]\right), \alpha_v = \mathrm{softmax}\left(s_v^{(k)}\right),$$
$$s_w^{(k)} = \sigma\left(\left[h_u^{(k-1)}; h_w^{(k-1)}; h_v^{(k-1)}\right]\right), \alpha_w = \mathrm{softmax}\left(s_w^{(k)}\right),$$

where $[h_v^{(k-1)}; h_u^{(k-1)}; h_w^{(k-1)}]$, $[h_u^{(k-1)}; h_v^{(k-1)}; h_w^{(k-1)}]$, and $[h_u^{(k-1)}; h_w^{(k-1)}; h_v^{(k-1)}]$ is a concatenation of vectors in $T_u(v), T_m(v)$ and $T_l(v)$, σ is $LeakyReLU$ activation function, α_u, α_v, α_w, $s_u^{(k-1)}$, $s_v^{(k-1)}$, and $s_w^{(k-1)} \in R^{d_{h_v} \times 1}$.

Our model concatenates vectors in triples $T_u(v)$, $T_m(v)$, and $T_l(v)$, and utilizes aggregation functions F_u, F_m, and F_l to generate node term-walk feature information from distinct positions. Subsequently, we assign weights α_u, $alpha_v$, and $alpha_w$ to the term-walk feature information of each node v for generating the final node embedding information. In the KTH iteration, the sets $T_u(v)$, $T_m(v)$, and $T_l(v)$ provide information ($m_{vu}^{(k)}$, $m_{vm}^{(k)}$, $m_{vl}^{(k)}$) for node v.

$$m_{vu}^{(k)} = \sum \frac{\alpha_u \cdot F_u([h_v^{(k-1)}; h_u^{(k-1)}; h_w^{(k-1)}])}{|T_u(v)|},$$
$$m_{vm}^{(k)} = \sum \frac{\alpha_v \cdot F_m([h_u^{(k-1)}; h_v^{(k-1)}; h_w^{(k-1)}])}{|T_m(v)|},$$
$$m_{vl}^{(k)} = \sum \frac{\alpha_w \cdot F_l([h_u^{(k-1)}; h_w^{(k-1)}; h_v^{(k-1)}])}{|T_l(v)|},$$

where $[;]$ denotes vector concatenation, $|T_u(v)|$, $|T_m(v)|$, and $|T_l(v)|$ represent the cardinality of triples in sets $T_u(v)$, $T_m(v)$, and $T_l(v)$ respectively. α_u, α_v, and α_w denote the contributions of different term-walk features of a node to its aggregation information.

Therefore, in the KTH iteration, the weighted node information $m_{vu}^{(k)}$, $m_{vm}^{(k)}$ and $m_{vl}^{(k)}$ from the sets $T_u(v)$, $T_m(v)$ and $T_l(v)$ can be summarized as

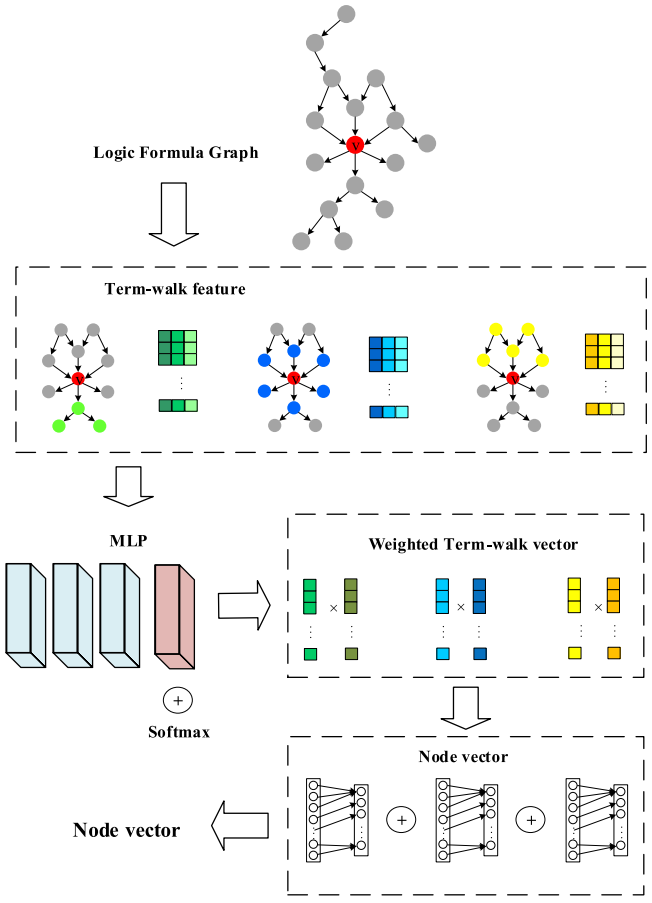

Fig. 5. Graph node information aggregation.

$m_v^{(k)}$:

$$m_v^{(k)} = m_{vu}^{(k)} + m_{vm}^{(k)} + m_{vl}^{(k)}.$$

4.5. Graph node information propagation

ASTGNNS updates the node vector $h_v^{(k)}$ with the total node information $m_v^{(k)}$ from the sets $T_u(v)$, $T_m(v)$ and $T_l(v)$ and the current state vector $h_v^{(k)}$ of node v.

$$h_v^{(k)} = h_v^{(k-1)} + F_{sum}(m_v^{(k)}),$$

where F_{sum} is the node information propagation function.

4.6. Graph aggregation

The graph classification task divides graph pooling into two categories: global pooling and hierarchical pooling. Global pooling performs a pooling operation on all nodes in the graph based on their characteristics. Hierarchical pooling is performed iteratively on all graph nodes according to their topology, gradually reducing the number of nodes until the final embedding vector is generated.

To preserve global and hierarchical characteristics, ASTGNNS combines self-attention pooling [42], global average pooling [43], and global maximum pooling to generate the ultimate formula graph embedding vector. After K iterations, ASTGNNS applies a self-attention-pooling operation (SAGPool) to obtain a new node representation from all nodes $h_v^{(k)}$ in the logical formula diagram. SAGPool employs a graph convolutional neural network approach to calculate

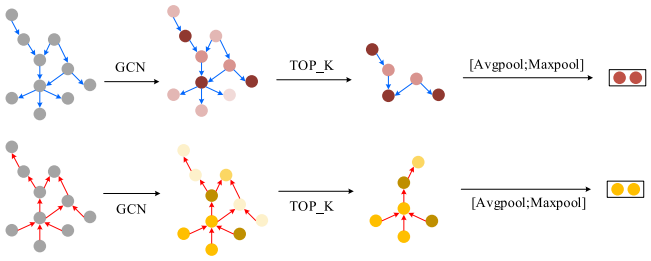

Fig. 6. Graph aggregation.

self-attention scores for each node, considering both forward and backward directions in bidirectional logical formula graphs.

$$Z = \sigma(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}h_v^{(k)}\Theta_{att}),$$
$$Z_{rev} = \sigma(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}_{rev}\widetilde{D}^{-\frac{1}{2}}h_v^{(k)}\Theta_{att}),$$

where Z and Z_{rev} are the node self-attention function based on the forward logical formula graph and the reverse logical formula graph, respectively, $\widetilde{A}$ is the adjacency matrix containing the self-ring, Θ_{att} is the only trainable parameter in SAGPool.

The model arranges nodes in a descending order based on their forward and reverse self-attention scores, subsequently selecting a specific proportion of nodes to reduce the graph's size.

$$idx = top\_rank(Z, [kN]), Z_{mask} = Z_{idx},$$
$$idx_{rev} = top\_rank(Z_{rev}, [kN]), Z_{mask\_rev} = Z_{idx\_rev},$$

where the return value of the top_rank function is the index of the top $[kN]$ nodes and $k \in (0, 1]$ is the pooling ratio.

After obtaining the index matrix, the model transforms the node features and the adjacency matrix accordingly to generate new node representations $h_{v\_out}^{(k)}$ and $h_{v\_out\_rev}^{(k)}$.

$$h_{v\_out}^{(k)} = h_{v_{idx,:}}^{(k)} \odot Z_{mask}, h_{v\_out\_rev}^{(k)} = h_{v_{idx\_rev}}^{(k)} \odot Z_{mask_{rev}},$$
$$A_{out} = A_{idx,idx}, A_{out\_rev} = A_{idx\_rev,idx\_rev},$$

where $h_{v\_idx,:}^{(k)}$ is the new feature matrix rearranged by $h_v^{(k)}$ according to the index, $h_{v\_out}^{(k)}$ and A_{out} are the corresponding node feature matrix and adjacency matrix after pooling, and $h_{v\_out\_rev}^{(k)}$ and A_{out_rev} correspond to the reverse logical formula graph.

The ASTGNNS model indicates that $h_{v\_out}^{(k)}$ and $h_{v\_out\_rev}^{(k)}$ perform global average pooling (AvgPool) and global maximum pooling (MaxPool) for this node. The final formula graph embedding vector h_G of $h_{v\_out}^{(k)}$ and $h_{v\_out\_rev}^{(k)}$ is the sum of joining together of the two global pools:

$$h_G = [AvgPool\{h_{v\_out}^{(k)}\}; MaxPool\{h_{v\_out}^{(k)}\}] + [AvgPool\{h_{v\_out\_rev}^{(k)}\}; MaxPool\{h_{v\_out\_rev}^{(k)}\}],$$

where $AvgPool$ takes the average of nodes on the graph, $MaxPool$ takes the maximum nodes on the graph, $h_G \in R^{d_{h_v}}$ (see Fig. 6).

4.7. Binary classifier

In the premise selection task, the binary classifier assesses the utility of the input premise embedding vector and conjecture embedding vector by assigning scores. The graph embedding vectors (h_p, h_c) representing candidate premises and conjectures are fed into the classification function F_{class} to obtain scores indicating the usefulness of candidate premises given a specific conjecture.

$$z = F_{class}([h_p; h_c]),$$

where $z \in \mathbb{R}^2$ is the score of how useful and useless the candidate premise is for the conjecture.

The binary classifier categorizes the candidate premises by standardizing the scores of their usefulness and uselessness for the conjecture. $\hat{y}$ represents the probability of a normalized premise being useful or useless.

$$\hat{y} = softmax(z), \hat{y}_i = \frac{exp(z_i)}{\sum_j exp(z_j)},$$

where z_i is the ith element in z.

4.8. Loss function

The premise selection problem is a classification problem, so the cross entropy loss of the predicted value $\hat{y}$ and the true value y is selected as its loss function $L(\hat{y}, y)$:

$$L(\hat{y}, y) = \sum_i (y_i ln(\hat{y}_i) + (1 - y_i) ln(1 - \hat{y}_i)),$$

where $y_i, \hat{y}$ are the values of the true value and the predicted value in the ith class, y is a one-hot encoding of the true value.

5. Experiments

The premise selection model in our study is implemented using the PyTorch framework. All experiments are conducted on an AMD 4029GP-TRT server with a Nvidia RTX 2080Ti GPU, 256G memory, and Center OS 7.6 Linux version. We provide detailed descriptions of the dataset, evaluation metrics, network configuration, and parameter settings used in our model. Furthermore, we evaluate the effectiveness of the premise selection model based on ASTGNNS through rigorous comparisons involving model performance, graph representation analysis, and ablation experiments.

5.1. Dataset

Based on the MPTP2078 question bank [44], we established the original dataset and the Conjunctive Normal Form (CNF) dataset. In propositional logic, the CNF of first-order formulas is analogous to that of propositional logic. Firstly, logical formulas are transformed into first-order formulas without existential quantifiers and implication symbols by eliminating implication connectives, shifting negation symbols, variable normalization, and Skolem normalization. Then, propositional logic's associative and distributive laws are employed to generate the final conjunctive normal form of first-order logic. The CNF dataset in this paper comprises the first-order logical formulae from the MPTP2078 question bank transformed by these operations.

We evaluate the experimental results of an ASTGNNS-based premise selection model on these two datasets, which consist of 1469 conjectures and 24087 premises used to prove them. These premises are paired with conjectures (each pair contains a "+" or "−" sign indicating whether they are useful or not). We construct triples (premise, conjecture, label) for each premise-conjecture pair, forming a training/validation/testing dataset. A premise selection model is trained on this dataset, where each premise is a candidate for a given conjecture. The label represents binary classification with values 0 or 1 (a label of 1 indicates usefulness while 0 denotes non-usefulness).

The MPTP2078 formula (premise and conjecture) is presented linearly, corresponding to the same order in the Mizar. Each conjecture is associated with an extensive set of premises, but only a small fraction of these premises are utilized to prove the conjecture. In this study, irrelevant premises are negatively sampled, while both premises and conjectures are represented using artificial features [5] based on symbols and sub-terms of first-order logical formulas. To roughly rank the premises within a given conjecture, we employ the K-Nearest Neighbor (KNN) algorithm [45]. Both datasets used in this research consist of equal samples, totaling 69,054 instances. For training, validation, and testing purposes, 80%, 10%, and 10%, respectively, were allocated (40,996 samples for training, 13,990 samples for validation, and 14,068 samples for testing). Refer to Table 2 for detailed information regarding these two datasets.

Table 2
Distribution of MPTP and CNF balanced datasets.

MPTP (CNF)	Training	Valid	Test
Positive	27 663	3417	3432
Negative	27 556	3485	3471
Total	55 219	6902	6903

Table 3
Parameter settings.

Model parameter	Settings
Node vector dimension d_{h_v}	128, 256, 512
Iterations K	1, 2
Training rounds	100, 150, 200
Learning rate	0.01
Learning rate decay coefficient	0.1
Batch size	16, 32, 64

5.2. Experiment settings

The parameters of our premise selection model based on ASTGNNS are configured as follows: (1) The model is trained using the default settings of the adaptive moment estimation Adam optimizer [41]; (2) The batch size is set to 32; (3) A regularization parameter of 0.0001 is employed; (4) An initial learning rate of 0.01 is utilized; (5) The ReduceLROnPlateau strategy from the PyTorch library is adopted for automatic learning rate adjustment. Table 3 presents a comprehensive overview of all premise selection model parameter configurations based on ASTGNNS.

Our proposed ASTGNNS iterates $\frac{1}{2}$ times on different node vector dimensions (128, 256, 512). The model is then tested on the validation set, and the corresponding results are saved after each training round. After 100 rounds, the model selects the round with the lowest loss on the validation set for testing on the test set and outputs the final classification accuracy.

Four indices, including Accuracy, Recall, Precision, and F_1, were selected to assess the classification performance of the premise selection model based on ASTGNNS.

$$Accuracy = \frac{TP + TN}{Total}, Recall = \frac{TP}{TP + FN},$$
$$Precision = \frac{TP}{TP + FP}, F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall},$$

where TP is the number of positive samples with correct classification in the classification model, TN is the number of negative samples with correct classification in the classification model, FN is the number of negative samples with classification errors in the classification model, FP is the number of positive samples with classification errors in the classification model, and $Total$ is the number of all samples in the data set. The higher the Accuracy, Recall, Precision, and F_1 indexes, the better the model classification effect.

A higher accuracy, recall, precision, and F_1 score indicate a better classification effect for the model. In ASTGNNS, with a node vector dimension of 256 and one iteration, optimal results are achieved on the MPTP test set. Figs. 7 and 8 illustrate the prediction accuracy and loss of the AS-TWGNN model for one and two iterations on both training and validation sets when using node vector dimensions of 128, 256, or 512.

We employed several critical hyperparameters in our experiments, such as node vector dimensions, number of iterations, training epochs, learning rate, and batch size. These hyperparameters are crucial in determining the model's complexity and generalization ability. We implemented the ReduceLROnPlateau learning rate decay strategy to reduce overfitting during training, automatically lowering the learning rate when the validation performance stagnates, thus promoting smoother convergence. Additionally, we introduced a regularization term with a coefficient of 0.0001 to further mitigate the risk of overfitting. The integration of these strategies ensures strong generalization across various datasets while effectively preventing overfitting.

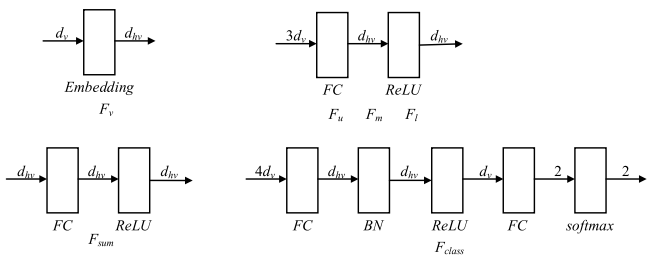

Fig. 7. Configuration of ASTGNNS in the premise selection.

5.3. Network configurations

A d_v-dimensional space represents the initial one-hot vector for each node shown in Fig. 7. F_V denotes an embedding network that transforms the initial one-hot vector into a node's initial state vector, which resides in a d_{h_v}-dimensional space. The configurations of F_u, F_m, and F_l are identical, consisting of Fully-Connected Layers (FC) with an input dimension of $3d_{h_v}$ and an output dimension of d_{h_v}. Similarly, the configuration of F_{sum} resembles that of F_u but only changes the input dimension to be equal to d_{h_v}. On the other hand, F_{class} comprises two fully connected layers: the first layer is an FC layer with a dimensionality of d_{h_v} followed by Batch Normalization (BN). In contrast, the second layer is an FC layer with a dimensionality of 2 achieved through the softmax function. It should be noted that here, we set $d_v = 793$, representing 793 node tokens uniformly denoted as "Var".

5.4. Experimental results and analysis

We compare the premise selection model based on ASTGNNS with mainstream GNNs and other baseline methods to show its advantages on the premise selection task, i.e., Graph convolutional neural network (GCN) [46], Graph attention neural network (GAT) [47], Graph sampling aggregation neural network (GraphSAGE) [48], Simplified graph convolutional neural network (SGC) [49] and Chebyshev spectral convolutional neural network (Cheb) [50]. Other baseline methods: Graph isomorphism network (GIN) [51], Graph Transformer (GT) [52], PC-GCN [26], TW-GNN [53], etc. To our knowledge, TW-GNN is the most performant GNN model for premise selection.

To ensure the validity of the comparative results, we re-run the GNN method within a unified premise selection model framework. During the experiment, only the components related to the GNN and the logical formula's graph representation are modified. We evaluate the premise selection model based on ASTGNNS on the MPTP and CNF datasets. We can conclude the following conclusions from Table 4.

- The premise selection methods based on GNN models can achieve high classification accuracy, while the mainstream GNN models generally exhibit suboptimal performance. For instance, GCN, GAT, and SGC models achieve Accuracy indicators of 86.25%, 85.38%, and 85.67%, respectively, on the original dataset. This indicates that mainstream GNNs primarily capture the topological structure of logical formula graphs but fail to capture deeper information within the logical formulas.
- Among other baseline methods, hand-designed feature-based GNNs PC-GCN and TW-GNN also suffer from similar limitations as mainstream GNNs do. For example, under the CNF dataset, PC-GCN and TW-GNN models attain an F_1 index of 83.98% and 83.72%, respectively, due to their inability to effectively aggregate information from distant or neighboring nodes.
- Our ASTGNNS-based premise selection model consistently outperforms other state-of-the-art GNN-based premise selection models regarding classification accuracy across various datasets. Specifically, compared to mainstream GNNs under the MPTP dataset, ASTGNNS achieves at least a 2% improvement; furthermore, it surpasses other baseline methods by 0.5% under the CNF dataset with a remarkable enhancement of approximately 3% compared to mainstream GNNs.

The attention mechanism's use in adjusting the GNN demonstrates its enhanced capacity to represent syntactic and semantic information about first-order logical formulas. Moreover, the graph representation of these formulas influences the model's classification performance. In this paper, our proposed model captures information from distant nodes and addresses concerns related to overfitting by simplifying node embedding information, thereby improving its overall fitting capability.

We explore the impact of different graph representations of first-order logical formulas on premise selection models. From Table 4, we also can conclude:

- Simplified DAGs can yield improved results in most premise selection tasks based on GNNs. For instance, when applied to the CNF dataset, GraphSAGE, SGC, and ASTGNNS exhibit an increase of 0.57%, 0.85%, and 0.26% in accuracy indicators, respectively, by utilizing simplified DAGs. This demonstrates that simplifying the graph representation reduces graph size without significant loss of information from DAGs, thereby enabling its utilization in premise selection models based on GNNs.
- The performance enhancement observed in certain GNN models using simplified DAGs may be attributed to their partial simplification, limiting their ability to learn information from logical formula graphs effectively.
- Existing representations of first-order logical formulas fail to fully capture both syntactic and potential semantic information inherent within them. Although DAGs offer a complete representation of logical formulas, unimportant components within these formulas can introduce errors into the final output formula graph vector. This could explain why simplifying the logical formula graph improves classification accuracy in premise selection tasks from an application view.

5.5. Presentation of training process

The training results of the ASTGNNS model with different parameter configurations are presented. Only the accuracy and loss are selected to demonstrate the model's performance under various parameters. As shown in Figs. 8 and 9, increasing the model dimension and iteration number enhances its fitting ability. However, excessively high values for these parameters may result in overfitting. It is observed that models with higher dimensions can be fitted more quickly during training, but their training speed significantly decreases as both dimensionality and iterations increase. ASTGNNS consistently achieves superior classification accuracy in premise selection tasks regardless of parameter settings.

5.6. Ablation experiment

The premise selection model based on ASTWGNNS comprises three key modules: first-order logic formula graph representation, graph neural network node information aggregation, and graph pooling. In this section, we designed eight ASTWGNNS-based premise selection model variants to evaluate the impact of different components. Specifically, ASTWGNNS(-t) removes the item walk feature, ASTWGNNS(-a) eliminates the attention mechanism, ASTWGNNS(-r) excludes the graph representation modification, ASTWGNNS(-p) deletes the attention pooling, ASTWGNN(-a-p) removes both the attention mechanism and attention pooling, ASTWGNN(-p-r) eliminates both the attention pooling and graph representation simplification, ASTWGNNS(-a-r) deletes

Table 4
Experimental results of model comparison for MPTP and CNF dataset.

Model	MPTP				CNF			
	Acc	Recall	Pre	F_1	Acc	Recall	Pre	F_1
	DAGs							
GCN [46]	86.25%	85.47%	86.69%	86.08%	80.51%	78.73%	81.44%	80.07%
GAT [47]	85.38%	85.56%	85.11%	85.34%	81.80%	81.25%	81.99%	81.62%
GraphSAGE [48]	86.15%	85.01%	86.86%	85.92%	82.96%	81.62%	83.69%	82.64%
SGC [49]	85.67%	85.12%	85.92%	85.52%	80.08%	79.86%	80.05%	79.95%
Cheb [50]	86.14%	85.21%	86.68%	85.94%	81.32%	79.08%	82.60%	80.80%
GIN [51]	85.93%	85.39%	86.19%	85.79%	82.09%	80.36%	83.08%	81.70%
GT [52]	86.16%	85.89%	86.24%	86.06%	82.33%	80.98%	83.06%	82.01%
PC-GCN [26]	87.77%	88.64%	87.01%	87.82%	84.17%	83.40%	84.56%	83.98%
TW-GNN [53]	88.12%	86.95%	88.15%	88.04%	84.24%	81.42%	86.15%	83.72%
ASTGNNS	**88.24%**	**87.76%**	**88.48%**	**88.12%**	**84.35%**	**82.50%**	**85.48%**	**83.97%**
	Simplified DAGs							
GCN	86.57%	86.26%	86.66%	86.46%	80.97%	78.91%	82.12%	80.48%
GAT	86.04%	85.65%	86.18%	85.91%	81.80%	80.52%	82.47%	81.48%
GraphSAGE	86.09%	85.68%	86.26%	85.97%	83.53%	83.02%	83.70%	83.36%
SGC	86.11%	85.56%	86.37%	85.96%	80.93%	79.71%	81.54%	80.62%
Cheb	85.90%	84.63%	86.70%	85.65%	80.03%	78.85%	80.57%	79.70%
GIN	86.74%	86.53%	86.78%	86.65%	83.04%	82.41%	83.31%	82.86%
GT	86.74%	86.21%	87.02%	86.61%	82.86%	82.29%	83.09%	82.69%
PC-GCN	88.19%	87.38%	88.70%	88.04%	84.40%	84.73%	84.02%	84.37%
TW-GNN	88.29%	87.59%	88.43%	88.05%	84.64%	84.42%	84.65%	84.54%
ASTGNNS	**88.77%**	**88.67%**	**88.74%**	**88.70%**	**85.17%**	**84.14%**	**85.75%**	**84.94%**

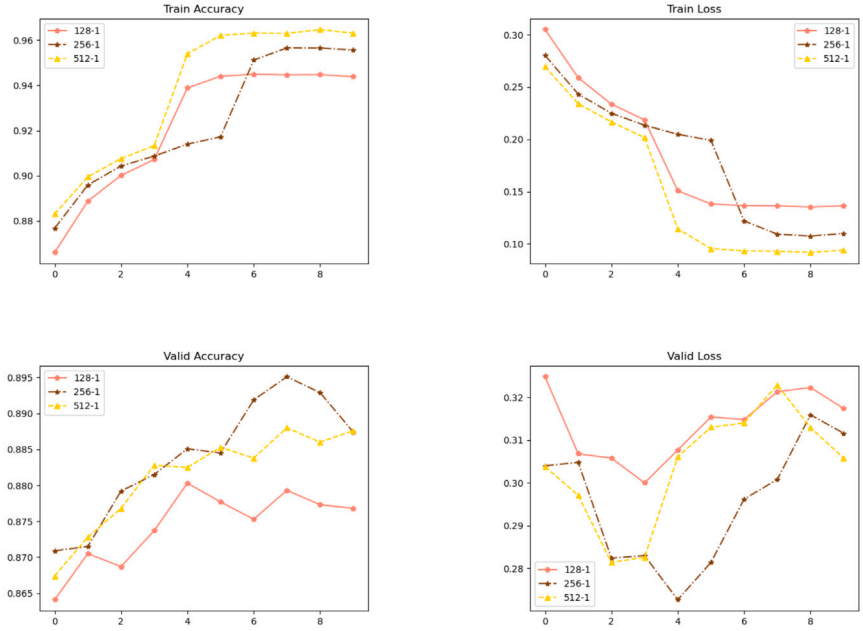

Fig. 8. Model prediction accuracy and prediction loss for ASTWGNN with 1 iterations.

Table 5
Ablation experiments on MPTP and CNF datasets.

Model	MPTP		CNF	
	Acc	F_1	Acc	F_1
ASTGNNS (-r)	88.58%	88.59%	84.35%	83.97%
ASTGNNS (-a)	88.34%	88.13%	84.88%	84.89%
ASTGNNS (-a-r)	88.47%	88.40%	84.75%	84.78%
ASTGNNS (-p)	88.28%	88.18%	84.84%	84.59%
ASTGNNS (-p-r)	87.70%	87.47%	84.85%	84.76%
ASTGNNS (-a-p)	88.29%	88.05%	84.64%	84.54%
ASTGNNS (-a-p-r)	88.12%	88.04%	84.24%	83.72%
ASTGNNS (-t)	86.04%	85.91%	81.80%	81.62%
ASTGNNS	**88.77%**	**88.70%**	**85.17%**	**84.94%**

both the attention mechanism and simplified graph representation, and ASTWGNNS(-a-p-r) removes the graph representation simplification, attention mechanism, and attention pooling. The ablation study results are presented in Table 5.

(1) According to Table 5, the classification performance of AS-TWGNNS(-t) is suboptimal. This suggests that item walk features play a critical role in the premise selection task, and incorporating an item walk graph neural network can significantly enhance model performance.
(2) The performance of ASTWGNNS(-a-p-r) ranks below that of ASTWGNNS(-r), ASTWGNNS(-a), and ASTWGNNS(-p). This indicates that graph simplification representation, attention mechanism, and attention pooling all substantially impact the classification ability of the premise selection model. Furthermore, on

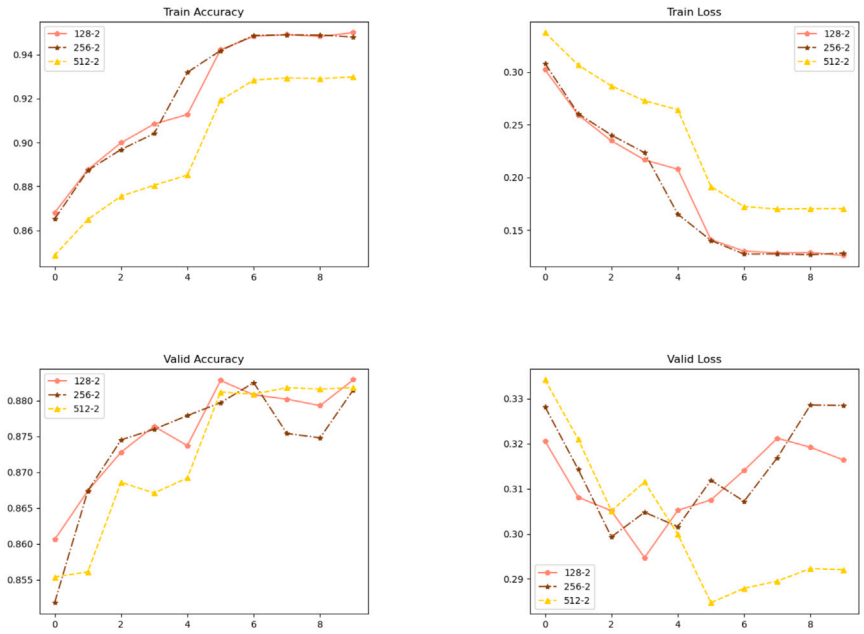

Fig. 9. Model prediction accuracy and prediction loss for ASTWGNN with 2 iterations.

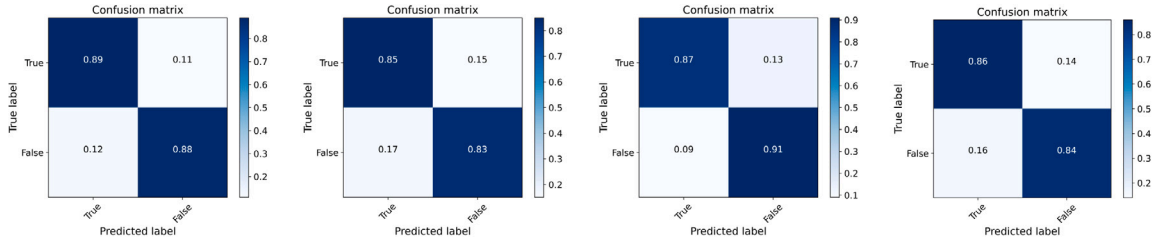

Fig. 10. Confusion matrix of the ASTGNNS.

the MPTP dataset, the classification accuracy of ASTWGNNS(-p-r) is lower than that of ASTWGNNS(-a-p-r), suggesting that modifying a single module does not necessarily improve the model's classification performance.

(3) The ASTWGNNS model consistently outperforms other variants in the premise selection task. This demonstrates that combining graph simplification representation, attention mechanism, and attention pooling yields superior classification results, and removing any component compromises the overall performance.

5.7. Experiment discussion

Fig. 10 shows the confusion matrices of the ASTGNNS model when it was experimented on the datasets MPTP and CNF using two graph representations: DAGs and Simplified DAGs. The TP and TN values of the confusion matrices of the ASTGNNS model are very high under different datasets and different graph representation methods, demonstrating that the model has excellent performance.

In Fig. 11, we visually compared the performance of various graph neural network models using a CD diagram, which illustrates the accuracy of different models on the FOF (left) and CNF (right) datasets in a tree-like structure, providing an intuitive hierarchical comparison of their relative strengths and weaknesses. Each node represents a model, and it is evident that the ASTGNNS model ranks first on both datasets, demonstrating a significant performance gap compared to most other models. Fig. 12 compares accuracy rates between ASTGNNS and other models on different datasets. It is visible from the graphs that ASTGNNS demonstrates outstanding performance on both datasets.

Fig. 13 shows the impact of different graph representation methods on the performance of the premise selection model, that is, the performance of different graph representations under multiple graph neural network models. It can be seen from the figure that the accuracy of most models increases after adopting Simplified DAGs, indicating that this graph representation method can play a role in the premise selection task. However, in a few models, the Simplified DAGs method reduces the accuracy, which may be related to the structural design of the model itself.

6. Conclusion

We propose simplified DAGs by eliminating duplicate quantifiers to capture first-order logical formulas' semantic and syntactic properties. Additionally, we define graph similarity based on the common path kernel function to demonstrate the rationality of these simplified DAGs. Leveraging these simplified DAGs and term-walk features, we introduce a novel Term-Walk GNN with attention mechanism and attention pooling (ASTGNNS). This model utilizes weighted term-walk feature information to generate node embedding vectors that preserve more semantic and syntactic properties of first-order logical formulas. Subsequently, an attention pooling technique is employed to obtain the final graph embedding vector. Experimental results on two datasets validate our proposed model's superior accuracy in premise selection tasks. However, our study has several caveats. First, the experiments should be extended to more scenarios besides premise selection. Second, more simplified graph representations should be proposed, such as deleting repeated logical symbols, and merging the same type of symbols. In future work, we will propose additional graph representations that

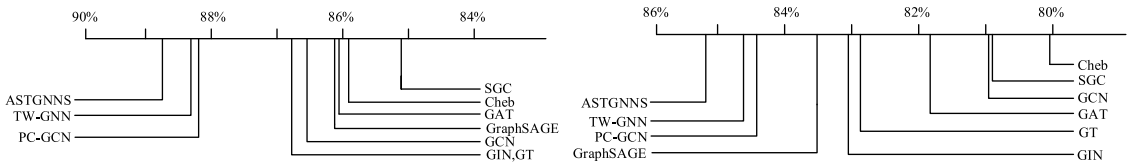

Fig. 11. A comparison of the performance of different graph neural network models.

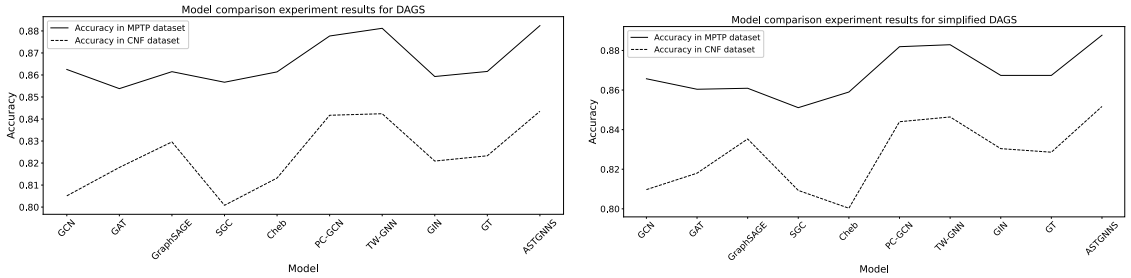

Fig. 12. Model comparison experiment results for DAGs and simplified DAGs.

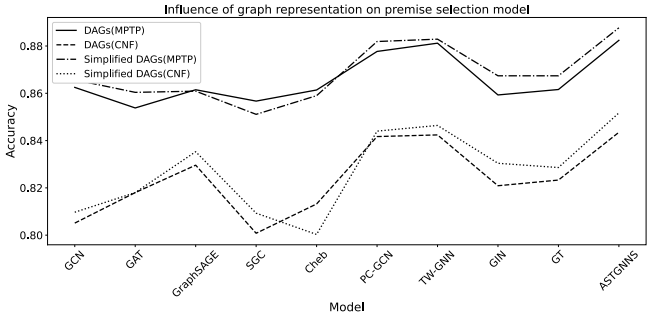

Fig. 13. A comparison of the performance of different graph neural network models.

retain information from first-order logical formulas while exploring diverse evaluation methods for different graph representations.

CRediT authorship contribution statement

Xingxing He: Writing – original draft, Supervision, Methodology, Funding acquisition, Conceptualization. **Zhongxu Zhao:** Writing – review & editing, Validation, Software, Methodology, Formal analysis. **Yongqi Lan:** Writing – original draft, Validation, Software, Conceptualization. **Yingfang Li:** Writing – review & editing, Visualization, Formal analysis. **Li Zou:** Writing – review & editing, Visualization, Formal analysis. **Jun Liu:** Writing – review & editing, Visualization, Formal analysis. **Luis Martínez:** Writing – review & editing. **Tianrui Li:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is supported by the National Natural Science Foundation of China (Grant No. 62176142), the Grant from MOE (Ministry of Education in China) Project of Humanities and Social Sciences (Grant No. 20XJCZH016), the Science and Technology Support Project of Sichuan province (Grant No. 2024YFHZ0316) and the Fundamental Research Funds for the Central Universities (Grant No. 2682024ZTPY041).

Data availability

I have share the link to the data/code related.

References

[1] T. Le Sergent, SCADE: A comprehensive framework for critical system and software engineering, in: SDL 2011: Integrating System and Software Modeling, 2012, pp. 2–3.
[2] S. Eggersgluss, R. Drechsler, Efficient data structures and methodologies for SAT-based ATPG providing high fault coverage in industrial application, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 30 (9) (2011) 1411–1415.
[3] A. Darwiche, Compiling knowledge into decomposable negation normal form, in: IJCAI, vol. 99, 1999, pp. 284–289.
[4] D. Jackson, I. Schechter, H. Shlyahter, Alcoa: the alloy constraint analyzer, in: Proceedings of the 22nd International Conference on Software Engineering, 2000, pp. 730–733.
[5] C. Kaliszyk, J. Urban, Mizar 40 for mizar 40, J. Automat. Reason. 55 (3) (2015) 245–256.
[6] I. Abdelaziz, M. Crouse, B. Makni, V. Austel, C. Cornelio, S. Ikbal, P. Kapanipathi, N. Makondo, K. Srinivas, M. Witbrock, A. Fokoue, Learning to guide a saturation-based theorem prover, IEEE Trans. Pattern Anal. Mach. Intell. 45 (1) (2023) 738–751.
[7] J. Meng, L.C. Paulson, Lightweight relevance filtering for machine-generated resolution problems, J. Appl. Log. 7 (1) (2009) 41–57.
[8] J. Zhang, J. Tian, P. Yan, S. Wu, H. Luo, S. Yin, Multi-hop graph pooling adversarial network for cross-domain remaining useful life prediction: A distributed federated learning perspective, Reliab. Eng. Syst. Saf. 244 (2024) 109950.
[9] J. Zhang, J. Tian, A.M. Alcaide, J.I. Leon, S. Vazquez, L.G. Franquelo, H. Luo, S. Yin, Lifetime extension approach based on the Levenberg–Marquardt neural network and power routing of DC–DC converters, IEEE Trans. Power Electron. 38 (8) (2023) 10280–10291.
[10] H. Wang, H. Luo, X. Qiao, M. Huo, X. Xu, Data-driven distributed robust monitoring and control optimization for interconnected systems, IEEE Trans. Ind. Inform. (2024).
[11] P. Yan, W. Gong, M. Li, J. Zhang, X. Li, Y. Jiang, H. Luo, H. Zhou, TDF-net: Trusted dynamic feature fusion network for breast cancer diagnosis using incomplete multimodal ultrasound, Inf. Fusion 112 (2024) 102592.
[12] G. Irving, C. Szegedy, A.A. Alemi, N. Een, F. Chollet, J. Urban, DeepMath - deep sequence models for premise selection, in: Proceedings of the Conference and Workshop on Neural Information Processing Systems, NeurIPS, vol. 29, 2016, pp. 2235–2243.
[13] L. Sarah, I. Geoffrey, S. Christian, K. Cezary, Deep network guided proof search, in: T. Eiter, D. Sands (Eds.), 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR-21, vol. 46, 2017, pp. 85–105.
[14] K. Chvalovsky, J. Jakubuv, M. Suda, J. Urban, ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E, in: Proceedings of the 27th International Conference on Automated Deduction, CADE 27, 2019, pp. 197–215.
[15] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Netw. 20 (1) (2008) 61–80.

[16] M. Shi, Y. Tang, X. Zhu, Y. Zhuang, M. Lin, J. Liu, Feature-attention graph convolutional networks for noise resilient learning, IEEE Trans. Cybern. 52 (8) (2022) 7719–7731.

[17] M. Wang, Y. Tang, J. Wang, J. Deng, Premise selection for theorem proving by deep graph embedding, in: Proceedings of the Conference and Workshop on Neural Information Processing Systems, NeurIPS, 2017, pp. 2786–2796.

[18] C. Gao, J. Zhu, F. Zhang, Z. Wang, X. Li, A novel representation learning for dynamic graphs based on graph convolutional networks, IEEE Trans. Cybern. 53 (6) (2023) 3599–3612.

[19] M. Schlichtkrull, T.N. Kipf, P. Bloem, R. van den Berg, I. Titov, M. Welling, A. Gangemi, R. Navigli, M.-E. Vidal, P. Hitzler, R. Troncy, L. Hollink, A. Tordai, M. Alam, Modeling relational data with graph convolutional networks, in: The 15th International Conference on the Semantic Web, 2018, pp. 593–607.

[20] Q. Lin, J. Liu, L. Zhang, Y. Pan, X. Hu, F. Xu, H. Zeng, Contrastive graph representations for logical formulas embedding, IEEE Trans. Knowl. Data Eng. 35 (4) (2023) 3563–3574.

[21] G. Huang, Z. Liu, L. Van Der Maaten, K.Q. Weinberger, Densely connected convolutional networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2017, pp. 4700–4708.

[22] B. Schölkopf, A.J. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, MIT Press, 2002.

[23] C.H. Elzinga, H. Wang, Kernels for acyclic digraphs, Pattern Recognit. Lett. 33 (16) (2012) 2239–2244.

[24] M. Crouse, I. Abdelaziz, C. Cornelio, V. Thost, L. Wu, K. Forbus, A. Fokoue, Improving graph neural network representations of logical formulae with subgraph pooling, 2019, arXiv preprint arXiv:1911.06904.

[25] J. Jakubův, J. Urban, ENIGMA: efficient learning-based inference guiding machine, in: H. Geuvers, M. England, O. Hasan, F. Rabe, O. Teschke (Eds.), Intelligent Computer Mathematics: 10th International Conference, CICM 2017, 2017, pp. 292–302.

[26] A. Paliwal, S. Loos, M. Rabe, K. Bansal, C. Szegedy, Graph representations for higher-order logic and theorem proving, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2020, pp. 2967–2974.

[27] Y. Xie, Y. Liang, M. Gong, A.K. Qin, Y.-S. Ong, T. He, Semisupervised graph neural networks for graph classification, IEEE Trans. Cybern. 53 (10) (2023) 6222–6235.

[28] A. Darwiche, P. Marquis, A knowledge compilation map, J. Artificial Intelligence Res. 17 (2002) 229–264.

[29] M. Rawson, G. Reger, Directed graph networks for logical reasoning, in: Proceedings of the 7th Workshop on Practical Aspects of Automated Reasoning, 2020, pp. 109–119.

[30] E. Aygün, A. Anand, L. Orseau, X. Glorot, S.M. Mcaleer, V. Firoiu, L.M. Zhang, D. Precup, S. Mourad, Proving theorems using incremental learning and hindsight experience replay, in: K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, S. Sabato (Eds.), Proceedings of the 39th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 162, PMLR, 2022, pp. 1198–1210.

[31] A. Fokoue, I. Abdelaziz, M. Crouse, S. Ikbal, A. Kishimoto, G. Lima, N. Makondo, R. Marinescu, An ensemble approach for automated theorem proving based on efficient name invariant graph neural representations, in: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, 2023, pp. 3221–3229.

[32] M. Mikuła, S. Antoniak, S. Tworkowski, B. Piotrowski, A. Jiang, J.P. Zhou, C. Szegedy, Ł. Kuciński, P. Miłoś, Y. Wu, Magnushammer: A transformer-based approach to premise selection, in: The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23, 2023.

[33] A. Bauer, M. Petković, L. Todorovski, MLFMF: data sets for machine learning for mathematical formalization, Adv. Neural Inf. Process. Syst. 36 (2024).

[34] S. Lamont, M. Norrish, A. Dezfouli, C. Walder, P. Montague, BAIT: Benchmarking (embedding) architectures for interactive theorem-proving, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, 2024, pp. 10607–10615, 9.

[35] K. Yang, A. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R.J. Prenger, A. Anandkumar, Leandojo: Theorem proving with retrieval-augmented language models, Adv. Neural Inf. Process. Syst. 36 (2024).

[36] Z. Li, J. Sun, L. Murphy, Q. Su, Z. Li, X. Zhang, K. Yang, X. Si, A survey on deep learning for theorem proving, 2024, arXiv preprint arXiv:2404.09939.

[37] Q. Liu, Y. Xu, X. He, Attention recurrent cross-graph neural network for selecting premises, Int. J. Mach. Learn. Cybern. 13 (5) (2022) 1301–1315.

[38] B. Bollobás, Modern Graph Theory, vol. 184, Springer Science & Business Media, 1998.

[39] R.B. Bapat, Graphs and Matrices, vol. 27, Springer, 2010.

[40] W. Imrich, S. Klavzar, D.F. Rall, Topics in Graph Theory: Graphs and Their Cartesian Product, CRC Press, 2008.

[41] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.

[42] J. Lee, I. Lee, J. Kang, Self-attention graph pooling, in: Proceedings of the 36th International Conference on Machine Learning, ICML, 2019, pp. 3734–3743.

[43] M. Lin, Q. Chen, S. Yan, Network in network, in: International Conference on Learning Representations, ICLR, 2014, pp. 1–10.

[44] A. Naumowicz, An experiment on mizar adjectives with extra visible arguments, in: 2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC, 2020, pp. 97–100.

[45] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE Trans. Inform. Theory 13 (1) (1967) 21–27.

[46] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, 2016, arXiv preprint arXiv:1609.02907.

[47] J. Gao, J. Gao, X. Ying, M. Lu, J. Wang, Higher-order interaction goes neural: A substructure assembling graph attention network for graph classification, IEEE Trans. Knowl. Data Eng. 35 (2) (2023) 1594–1608.

[48] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: Proceedings of the Conference and Workshop on Neural Information Processing Systems, NeurIPS, vol. 30, 2017.

[49] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, K. Weinberger, Simplifying graph convolutional networks, in: Proceedings of the 34th International Conference on Machine Learning, ICML, 2019, pp. 6861–6871.

[50] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, in: Proceedings of the Conference and Workshop on Neural Information Processing Systems, NeurIPS, vol. 29, 2016, pp. 3844–3852.

[51] C. Wu, Y. Lou, J. Li, L. Wang, S. Xie, G. Chen, A multitask network robustness analysis system based on the graph isomorphism network, IEEE Trans. Cybern. (2024).

[52] A. Gui, J. Ye, H. Xiao, G-adapter: Towards structure-aware parameter-efficient transfer learning for graph transformer networks, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, 2024, pp. 12226–12234, 11.

[53] Q. Liu, Y. Xu, Axiom selection over large theory based on new first-order formula metrics, Appl. Intell. 52 (2) (2022) 1793–1807.