Subgraph Search over Neural-Symbolic Graphs

Ye Yuan
Beijing Institute of Technology
yuan-ye@bit.edu.cn

Delong Ma
Northeastern University, China
madelong@stumail.neu.edu.cn

Anbiao Wu
Northeastern University, China
anbiaowu@stumail.neu.edu.cn

Jianbin Qin
Shenzhen University
qinjianbin@szu.edu.cn

ABSTRACT

In this paper, we propose *neural-symbolic graph databases* (NSGDs) that extends traditional graph data with *content and structural embeddings* in every node. The content embeddings can represent unstructured data (e.g., images, videos, and texts), while structural embeddings can be used to deal with incomplete graphs. We can advocate machine learning models (e.g., deep learning) to transform unstructured data and graph nodes to these embeddings. NSGDs can support a wide range of applications (e.g., online recommendation and natural language question answering) in social-media networks, multi-modal knowledge graphs and etc. As a typical search over graphs, we study subgraph search over a large NSGD, called *neural-symbolic subgraph matching* (NSMatch) that includes a novel ranking search function. Specifically, we develop a general algorithmic framework to process NSMatch efficiently. Using real-life multi-modal graphs, we experimentally verify the effectiveness, scalability and efficiency of NSMatch.

CCS CONCEPTS

• **Information systems** → **Graph-based database models**.

KEYWORDS

subgraph search; neural-symbolic; embedding

ACM Reference Format:
Ye Yuan, Delong Ma, Anbiao Wu, and Jianbin Qin. 2023. Subgraph Search over Neural-Symbolic Graphs. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '23), July 23–27, 2023, Taipei, Taiwan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3539618.3591773

1 INTRODUCTION

It is increasingly common to find real-life data modeled as graphs, which represent entities as nodes and relationships between entities as edges. Indeed, graphs have found prevalent use in online recommendation [20], question answering [7], link prediction [4, 39], information retrieval [38, 41], among other things. However, the existing graphs are represented with pure symbols denoted in the form of text, which weakens the capability of machines to describe and understand the real world. Therefore, it is necessary to ground symbols to corresponding images, sound and video data and map symbols to their corresponding referents with meanings in the physical world, enabling machines to generate similar experiences like a real human [18]. To realize this through graphs, we propose *neural-symbolic graph databases* (NSGDs).

Below we present examples of subgraph search over multi-modal knowledge graphs to illustrate the use-cases of NSGDs.

Example 1.1. **Subgraph search on multi-modal graphs.** IMGpedia [11], Richpedia [31] and YAGO15K [23] are real multi-modal knowledge graphs G in which a node may contain images and texts. Fig. 1 shows a part of these graphs illustrating information for movies (mo), such as the actors (ac) and director (di) of a movie. The movie node mo_1 in G has three pairs of attributes/values: (name, Titantic), (director, James Cameron) and (storyline, a long paragraph of texts). An edge (mo_1, ac_2) from mo_1 to an actor node ac_2 indicates a relation "hasActor". The node ac_2 also contains three pairs of attributes/values, one of which is the actor's photo. On the other hand, the knowledge graph is usually incomplete e.g., the red dotted arrow (mo_1, ac_3) does not appear in G.

In applications of online recommendation, question answering and information retrieval, a graph pattern could be issued over a knowledge graph. For example, a graph pattern P_1 in Fig.2 describes a movie as follows: a leading actor of the movie is Winslet; P_1 contains an unknown actor cooperating with Winslet in this movie but P_1 has his photo associated with him; and the answer to P_1 should return the name of the movie. Graph pattern P_2 in Fig.2 describes a football club as follows: P_2 has two photos of the club president (pr) and the team coach (co) but not their names; P_2 also includes a star player (pl) of this club with name Ronaldo; and P_2 should return the possible names of the club. Graph pattern P_3 in Fig. 2 is issued over multi-modal social networks, e.g., Twitter. □

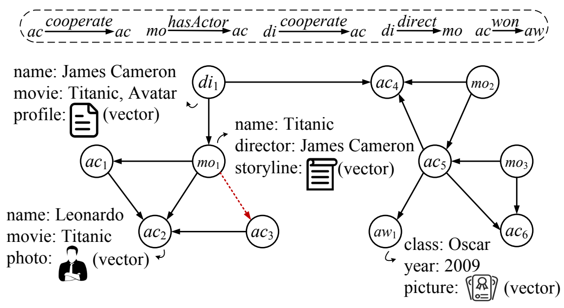


Figure 1: A part of multi-modal knowledge graph.

SIGIR '23, July 23–27, 2023, Taipei, Taiwan
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9408-6/23/07…$15.00
https://doi.org/10.1145/3539618.3591773

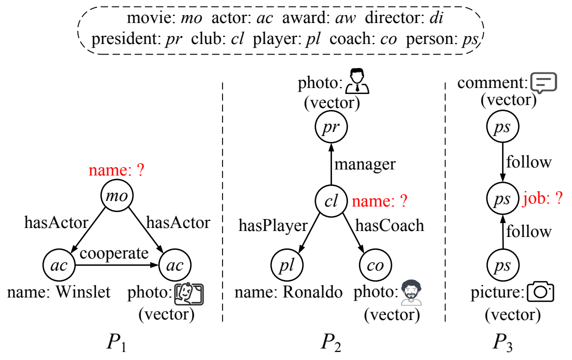


Figure 2: Graph patterns.

The example shows that both pattern and data are a combination of graph structure and unstructured data (e.g., image and text). The existing techniques fail to solve the novel problem. This example thus raises several questions: How should we define graph data models to support unstructured data and catch the incompleteness of graphs? If such a graph model exists, how to define the subgraph search over the model? Are the new problems for subgraph search harder than the traditional ones? Putting these together, above all, can we develop practical and efficient algorithms to process subgraph search over such large graph data?

Contributions & organization. This paper makes an effort to answer above questions, from foundation to practice.

(1) NSGD. We propose a novel graph data model–neural-symbolic graph database, referred to as NSGD (Section 2). An NSGD extends a property graph model G with content and structural embeddings in every node and edge of G. The content embeddings represent unstructured data (e.g., images, videos and texts in Fig. 1) that can be effectively transformed to the feature embeddings by machine learning models [22]. Many neural methods (e.g., TransE [3], ConE [44] and BetaE [27] convert nodes and edges into structural embeddings that can be used to solve the incompleteness of graphs. We also conduct *symbolic enhancement* on the neural methods to further increase their effectiveness.

(2) NSMatch. We define neural-symbolic subgraph matching (NSMatch) by performing top-k subgraph search using sophisticated ranking functions in a large NSGD (Section 2). Given a graph pattern Q, NSMatch decomposes it to a set of star patterns, and assembles top answers from individual star (Section 3). NSMatch generates top answers of stars in a monotonic decreasing order of matching score. This nice property makes it possible to apply monotonic ranked joins to produce the final top k answers for Q without losing completeness (Section 5). NSMatch needs to process the KNN embedding search for which graph-based indices [13, 24] are most efficient. However, the embeddings from an NSGD may be skewed indexed into communities with different cohesiveness. We thus optimize the graph-based indices by balancing the embedding distribution to accelerate the search (Section 4).

(3) Experiments. Using real-life multi-modal graphs, we empirically evaluate our algorithms (Section 6). We find the following. (a) NSMatch has always above 90% search accuracy in different settings, while its competitors have below 80% search accuracy in the same scenarios. (b) NSMatch is feasible on large graphs. It takes only one second on a graph of 500M nodes and 3000M edges and is scalable to different graph sizes. (c) NSMatch can support efficient cross-modal search, over knowledge graphs, that cannot be expressed with conventional subgraph search.

2 PROBLEM DEFINITION

We start with basic concepts of NSGD and then formalize the problem of NSMatch.

Neural-Symbolic Graph Database (NSGD). We consider directed labeled NSGDs, defined as $G = (V, E, L, F, T)$, where (1) V is a finite set of nodes and $E \subseteq V \times V$ is a set of directed edges; (2) each node $v \in V$ (resp. each edge $e \in E$) has a label $L(v)$ (resp. $L(e)$); (3) each node $v \in V$ carries a set $F(v) = \{A_i = a_i\}$ of attributes/values, where A_i is an attribute and a_i is a constant value; and (4) each node $v \in V$ carries a set T of two types of vectors: content vector $\mathbf{C}_v$ and structural vector $\mathbf{S}_v$, i.e., $T(v) = \{\mathbf{C}_v, \mathbf{S}_v\}$.

The content vector $\mathbf{C}_v$ is the embedding of the unstructured data, such as text, image or video associated with node v. $\mathbf{C}_v$ is a type of value in F. An NSGD can support multiple vectors which represent different modal data in node v. For a clear presentation, we assume that a node contains only one content vector. Given two structural vectors $\mathbf{S}_u$ and $\mathbf{S}_v$ of nodes u and v, $\mathbf{S}_u$ and $\mathbf{S}_v$ can be used to check whether there exists an edge (u, v) from u to v. Both content and structural vectors are generated offline by the state-of-the-art deep learning models.

Notice that most of labels, attributes and values are short categorical data or numerical data that are not necessarily represented by vectors as the heavy unstructured data.

Fig.1 gives an NSGD G describing the multi-modal knowledge graph. Every node and edge in G have their labels, e.g., the label *director* for node di_1 and the label *direct* for the edge (di_1, mo_1). Every node has pairs of attributes/values, e.g., (name, James Cameron). Among attributes/values, $profile$ and *photo* are unstructured data which can be represented by content vectors $\mathbf{C}_v$ transformed by deep learning models. Structural vectors for nodes mo_1 and ac_3 can be used to verify whether the edge (mo_1, ac_3) exists in G.

Intuitively, an NSGD extends a data graph $G = (V, E, L, F)$ with content and structural vectors $T(v) = \{\mathbf{C}_v, \mathbf{S}_v\}$. Consequently, an NSGD can support symbolic and neural computations over G. The symbolic computation can be a traditional combinatorial search (e.g., subgraph, clique, core finding) or logical reasoning over G. The neural computation can be a content vector search over G or structural completions of G by structural vectors. The symbolic and neural computations can be combined to define novel search, such as neural-symbolic subgraph matching.

Graph pattern. A graph pattern is a directed graph defined as $Q = (V_q, E_q, L_q, C_q)$, where (1) V_q and E_q are a set of pattern nodes and pattern edges, respectively; (2) L_q is a labeling function such that for each node $v \in V_q$ and edge $e \in E_q$, $L_q(v)$ is a node label and $L_q(e)$ is an edge label; and (3) for each node in $v \in V_q$, $C_q(v)$ specifies a content vector which is the embedding of unstructured data carried with v.

For example, pattern P_1 in Fig. 2 is a graph. It has three nodes with two labels *movie* and *actor*. Specifically, node *ac* carries a content vector, i.e., the embedding of an actor's photo.

Similarity function. Given two vectors $\mathbf{x}$ and $\mathbf{y}$ with the same dimension, their similarity functions $\delta(\mathbf{x}, \mathbf{y})$ are commonly used similarity metrics, including Euclidean distance, inner product, cosine similarity, and Hamming distance, depend on applications.

Neural-symbolic subgraph matching (NSMatch). Given a graph pattern Q=(V_q, E_q, L_q, C_q) and an NSGD G=(V, E, L, F, T), a subgraph match $h(Q)$ of Q in G is a mapping h from Q to G such that (1) for each node $u \in V_q$, $L_q(u) = L(h(u))$; and (2) for each edge e=(u,v) $\in E_q$, $L_q(e) = L(h(u), h(v))$. The neural-symbolic subgraph matching needs to compute a *match score* $S(h(Q))$ between Q and $h(Q)$. Given a similarity function $\delta(\cdot)$, $S(h(Q))$ is computed as

$$S(h(Q)) = \sum_{v \in V_q} \delta(\mathbf{C}_v, \mathbf{C}_{h(v)})$$

where $\mathbf{C}_v$ and $\mathbf{C}_{h(v)}$ are the content vectors of nodes v and $h(v)$, respectively.

Note that structural vectors can help a pattern identify missed matches if missed edges are added to G. As an example, pattern P_1 in Fig. 2 can find a match $mo_1ac_2ac_3$ in G in Fig. 1 if edge (mo_1, ac_3) (red dotted arrow) is added to G.

Problem definition. Given Q, G and $S(\cdot)$, the neural-symbolic subgraph matching is to find the top-k subgraph matches $Q(G, k)$, such that for any match $h'(Q) \notin Q(G, k)$, for all $h(Q) \in Q(G, k)$, $S(h(Q)) \geq S(h'(Q))$.

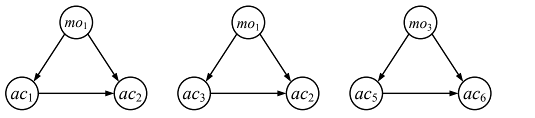


Figure 3: Top-3 subgraph matches of P_1 in G.

Example 2.1. One may issue a pattern P_1 in Fig.2 against the NSGD G in Fig.1 and want to find top-3 answers. We first compute all subgraph matches of P_1 in G: $mo_1ac_1ac_2$, $mo_1ac_3ac_2$, $mo_3ac_5ac_6$ and $mo_2ac_5ac_4$. The top-3 answers are $mo_1ac_1ac_2$, $mo_1ac_3ac_2$ and $mo_3ac_5ac_6$ shown in Fig. 3, since their matching scores are larger than that of $mo_2ac_5ac_4$ based on the similarity function $\delta(\cdot)$. □

It is NP-hard to process the NSMatch, since its special case (i.e., subgraph isomorphism) is NP-complete [14].

3 FRAMEWORK OF SEARCH

In this section, we introduce the algorithmic framework to attack the hard problem of NSMatch.

(1) *Pattern decomposition.* Once a graph pattern Q is submitted, a procedure DecQ is invoked to decompose Q into a set of star patterns $SQ = \{sq_1, ...sq_m\}$.

(2) *Star matching.* In this step, we propose an algorithm SMat which can efficiently generate a set of top matches for each star pattern and guarantee that the matches are generated progressively in a descending order of the match score for each star pattern.

(3) *Top-k join.* The top matches produced by SMat for multiple star patterns are then joined by a join procedure JoinK. JoinK terminates once the top-k matches are identified, or there is no chance to obtain better matches.

In the rest of this paper, we first introduce star matching, and afterwards pattern decomposition and top-k join.

4 TOP-K STAR MATCHING

In this section, we propose the algorithm SMat for computing the top-k star matches. We first give a framework of SMat in the sub-section 4.1. Next, in the sub-section 4.2, we introduce how to efficiently identify the top-k matching nodes for any node of a star pattern. Finally, in the sub-section 4.3, we illustrate how to accurately complete the missing edges of top-k star matches.

4.1 Framework of SMat

```
Algorithm SMat
Input: a star pattern $sq$, an NSGD $G$, any content vectors $\mathbf{C_{sq}}$ and
       $\mathbf{C_g}$, any structural vector $\mathbf{S_g}$, integer $k$
Output: top-k match set $R$.
1.  initialize set $R = \emptyset$;
2.  initialize priority queue $P = \emptyset$;
3.  identify the pivot matches $V_p = \{v_p\}$ and every leaf node's
    matches $V_l = \{v_l\}$ in $G$;
4.  invoke EJud to judge whether the edge $(v_p, v_l)$ should exist
    by $\mathbf{S_g(v_p)}$ and $\mathbf{S_g(v_l)}$;
5.  add $(v_p, v_l)$ to $G$ if EJud returns true;
6.  invoke NMat to find the top-$h$ descending leaf matches
    $V_l^h = \{v_l^h\}$ in $G$ by calculating the similarity function $\delta(\cdot)$
    between $\mathbf{C_{sq}}$ and $\mathbf{C_g}$;
7.  every $v_p$ scans its neighbors in $G$ to identify its leaf
    matches $V_l^p$;
8.  every $v_p$ determines the top-1 star match pivoted at $v_p$;
9.  add the best $k$ star matches from all the top-1 matches to $P$;
10. sort the leaf matches in $V_l^h \cup V_l^p$ for these $k$ star matches;
11. while ($|R| < k$) do
12.     pop the best match $m$ (pivoted at $v_p$) from $P$;
        $R = R \cup m$;
13.     generate the next best match $m'$ pivoted at $v_p$;
14.     insert $m'$ to $P$;
15. return $R$ as $sq(G, k)$;
```

Figure 4: Algorithmic framework of star matching SMat.

Fig. 4 shows the framework of SMat, where the inputs of SMat are an integer k, a star pattern sq with content vector $\mathbf{C_{sq}}$, and an NSGD G with content and structural vectors $\mathbf{C_g}$ and $\mathbf{S_g}$. Its output is the top-k matches $sq(G, k)$ of the star pattern sq over G.

Notice that $\mathbf{C_{sq}}$, $\mathbf{C_g}$ and $\mathbf{S_g}$ denote the related vectors for any node in sq and G.

SqMat includes the following three phases.

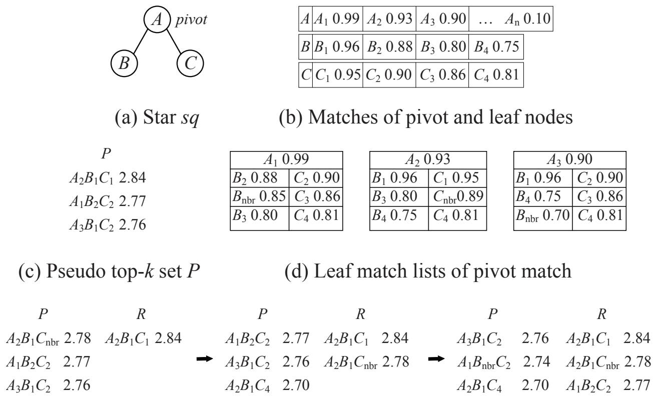
Phase 1. SMat first identifies candidate node matches V_p and V_l for the pivot node p and each leaf node l of sq in G (line 3). For each $v_p \in V_p$ and $v_l \in V_l$, SMat invokes EJud (given in Section 4.3) to judge whether an edge (v_p, v_l) should exist in G (line 4). SMat adds (v_p, v_l) to G if EJud returns true (line 5). This step can assure a completed set of star matches of sq in G.

Next, SMat selects top-h leaf matches $V_l^h = \{v_l^h\}$ for every leaf node with a descending order by calculating the similarity $\delta(\cdot)$ between $\mathbf{C_{sq}}$ and $\mathbf{C_g}$ (line 6). During the identification, SMat invokes NMat (given in Section 4.2) to obtain the top-h leaf matches, where h is an integer pre-determined in the system. After that, every pivot match v_p identifies its leaf matches V_l^p by scanning its neighbors in G, and v_p can easily obtain its top-1 star match after the scanning (lines 7–8).

Phase 2. Among the top-1 star matches at every v_p, SqMat selects the best k star matches to form a pseudo top-k star matches which are inserted in a priority queue P. Note that, these pseudo top-k star matches contain the potential pivot nodes in the final top k answers (line 9). For the k pivots in the pseudo top-k star matches, SqMat sorts the leaf matches $V_l^h \cup V_l^p$ for each pivot (line 10). During the sorting, SqMat adds the node in V_l^p to its position in V_l^h, since the nodes in V_l^h have been sorted. The leaf matches pivoted at v_p are sufficed for SMat to obtain the top-k star matches. Therefore, SMat only keeps top-k leaf matches for v_p.

Phase 3. SqMat maintains the priority queue P. SqMat pops up the best match m from P, and inserts it into an answer set R (line 12). For the pivot node in m, it generates the next best match m' which is inserted into P (lines 13–14). Specifically, SqMat retrieves every v_p's leaf match lists and selects the one with the smallest difference by subtracting the largest score in the list. This phase repeatedly until $|R| = k$. Finally SqMat outputs R as the answer set (line 15).

As mentioned above, the value of h is pre-determined in the system. Intuitively, the larger the h is chosen, the more efficiency will be. If h is large enough, V_l^h may cover many leaf matches in v_p's neighbors. Therefore, the number of remaining neighbors of v_p to be scanned and calculated will be small. In contrast, if h is very large, every pivot will take more time to obtain its neighbors in the top-h leaf matches. In the experiments, we will choose a proper h to optimize the whole search processing.



A	A_1 0.99	A_2 0.93	A_3 0.90	… A_n 0.10
B	B_1 0.96	B_2 0.88	B_3 0.80	B_4 0.75
C	C_1 0.95	C_2 0.90	C_3 0.86	C_4 0.81

(b) Matches of pivot and leaf nodes

P
$A_2B_1C_1$ 2.84
$A_1B_2C_2$ 2.77
$A_3B_1C_2$ 2.76

(c) Pseudo top-k set P

A_1 0.99		A_2 0.93		A_3 0.90	
B_2 0.88	C_2 0.90	B_1 0.96	C_1 0.95	B_1 0.96	C_2 0.90
B_{nbr} 0.85	C_3 0.86	B_3 0.80	C_{nbr} 0.89	B_4 0.75	C_3 0.86
B_3 0.80	C_4 0.81	B_4 0.75	C_4 0.81	B_{nbr} 0.70	C_4 0.81

(d) Leaf match lists of pivot match

P	R		P	R		P	R
$A_2B_1C_{nbr}$ 2.78	$A_2B_1C_1$ 2.84		$A_1B_2C_2$ 2.77	$A_2B_1C_1$ 2.84		$A_3B_1C_2$ 2.76	$A_2B_1C_1$ 2.84
$A_1B_2C_2$ 2.77		➡	$A_3B_1C_2$ 2.76	$A_2B_1C_{nbr}$ 2.78	➡	$A_1B_{nbr}C_2$ 2.74	$A_2B_1C_{nbr}$ 2.78
$A_3B_1C_2$ 2.76			$A_2B_1C_4$ 2.70			$A_2B_1C_4$ 2.70	$A_1B_2C_2$ 2.77

(e) Procedure of searching top-3 matches

Figure 5: Procedure of star matching.

Example 4.1. We demonstrate how SMat computes top-3 matches for the star pattern sq in Fig.5(a) through the three phases. During Phase 1, SMat first identifies the candidate node matches $V_A = \{A_1, ..., A_n\}$ of pivot A and leaf nodes candidates V_B and V_C. Assuming $h = 4$, it next selects top-4 leaf matches for every leaf node with a descending order given in Fig.5(b) for V_B^4 and V_C^4. After that, every pivot candidate $v_A \in V_A$ finds its neighbors in V_B^4 and V_C^4. For example, the pivot match A_3 in V_A finds its neighbors B_1, B_4, C_2, C_3 and C_4 as shown in Fig.5(d). To this end, every pivot match v_A can easily obtain its top-1 star matches.

Phase 2 starts. Among the top-1 star matches at every v_A, SMat selects the best three star matches to form a pseudo top-3 star matches given in Fig.5(c). SMat then sorts the leaf matches for each pivot in a descending list. To form the list, SMat needs to scan the neighbors (denoted by X_{nbr}) of pivot matches which are not in V_l^h. For example, in Fig.5(d), among A_1's neighbors SMat finds its neighbor $B_{nbr} = 0.85$ is larger than $B_3 = 0.80$, and thus inserts B_{nbr} to its descending list. Similarly, SMat also computes $C_{nbr} = 0.89$ for A_2 and $B_{nbr} = 0.70$ for A_3. At the final of Phase 2, pivot matches A_1, A_2 and A_3 hold the descending lists for leaf matches as given in Fig.5(d).

Fig. 5(e) shows the procedure of Phase 3. SMat first pops from priority queue P the best match $A_2B_1C_1$ inserted into R, which is the top-1 match pivoted at A_2. SMat then generates the next best match $A_2B_1C_{nbr}$ pivoted at A_2 and adds it into P. Following the same rule, SqMat terminates until P pops up $A_1B_2C_2$ and $|R| = 3$. Finally the answer is returned as $R = \{A_2B_1C_1, A_2B_1C_{nbr}, A_1B_2C_2\}$. □

4.2 Top-h Node Matching

As shown in the framework of SMat, NMat identifies the top-h node matches by calculating the similarity function $\delta(\mathbf{C}_q, \mathbf{C}_g)$. Usually G is very large with billions of nodes (vectors), and hence it is time-consuming to obtain the *exact order* of the node matches. To conquer the problem, NMat leverages Approximate Nearest Neighbors Search (ANNS) [32] to find the top-h node matches. In this subsection, we first introduce the existing ANNS, and then improve ANNS based on the features of content vectors in the NSGD.

(1) Approximate nearest neighbors search.

We first give the definitions for Nearest Neighbors Search (NNS) and ANNS.

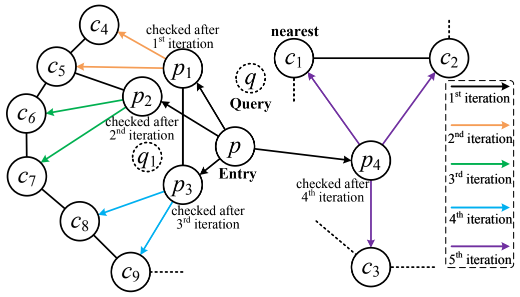
Nearest Neighbors Search (NNS). Given a query vector q and a finite set of vectors S in the Euclidean space E^d with dimension d, **NNS** obtains q's h-nearest neighbors (vectors) $\mathcal{R}$ by evaluating $\delta(x, q)$, where $x \in \mathcal{R}$ is described as follows:

$$\mathcal{R} = arg \min_{\mathcal{R} \subset S, |\mathcal{R}=h|} \sum_{x \in \mathcal{R}} \delta(x, q) \tag{1}$$

Approximate Nearest Neighbors Search (ANNS). Given a query vector q and a finite set of vectors S in the Euclidean space E^d with dimension d, **ANNS** builds an index $\mathcal{I}$ on S. It then gets a subset C of S by $\mathcal{I}$, and evaluates $\delta(x, q)$ to obtain the approximate h-nearest neighbors $\widetilde{\mathcal{R}}$ of the query vector q, where $x \in C$.

Due to its widespread adoption, ANNS has developed for a decade. Recently, with the introduction of the approximate linear time algorithm for constructing H-Nearest Neighbor Graph (HNNG) [8], researchers developed graph-based ANNS algorithms by constructing HNNG-like graph indices. Such graph indices have an extraordinary ability to express neighbor relationships, which make graph-based ANNS algorithms only need to visit fewer vectors in S to yield more accurate answers [12, 24]. So in this work, NMat advocates and further optimizes the graph-based ANNS algorithms for finding query q's top-h node matches.

An HNNG I_G can be constructed as follows [12]: every vector in S is connected to its h nearest vectors to form I_G in the Euclidean space E^d. A query process on I_G is executed as follows.



checked after 1st iteration
nearest
Query
checked after 2nd iteration
Entry
checked after 4th iteration
checked after 3rd iteration
1st iteration
2nd iteration
3rd iteration
4th iteration
5th iteration

Figure 6: Procedure of ANNS on index I_G.

Assume that graph in Fig. 6 is a built HNNG I_G. For a given query node (vector) q in I_G, NMat first randomly selects a node p as the entry node, follows its out-edges to reach p's neighbors, and chooses one to proceed following a principle that minimizes the distance to q. After that a new iteration starts from the chosen node. Colored arrows in Fig 6 show the complete query procedure from the entry node p to the query q's nearest neighbor c_1, which needs 5 iterations and 13 distance calculations.

Issues of ANNS on I_G. A social graph or knowledge graph G has many communities, e.g., users sharing the common sport hobby may form a community. Content vectors of nodes in a community of G may also constitute a community in the Euclidean space E^d [15, 35]. Based on the observation, we draw that nodes in the index I_G are unevenly distributed and form communities. These features lead to one serious issue of ANNS on I_G: the uneven distribution of nodes in the directions (e.g., north and south) of E^d. The uneven distribution in directions may lower the search efficiency over I_G. For example, it will take more iterations if q first searches a wrong direction with more nodes than other directions.

To solve the issue, NMat designs an edge deletion strategy to alleviate the problem caused by the uneven distribution.

(2) Edge deletion.

NMat performs the edge deletion as follows. (1) For any node u in I_G, NMat first sorts u's neighbors $\{v\}$ in an ascending order of the distance from u to v. Denote by E_u the edge set $E_u = \{(u,v) | v \text{ is a neighbor of } u \text{ in } I_G\}$. With the sorted E_u, NMat chooses the shortest edge $E_u[0]$ and marks it as checked. (2) NMat then judges whether there exists an unchecked edge $E_u[j]$ ($j > 0$) in E_u whose angle formed with v is less than a given α. If such $E_u[j]$ exists, the longer edge will be deleted. This deletion process is executed for all the remaining unchecked edges in E_u. (3) Denote by E'_u the the undeleted edge set in E_u after the above process. The next iteration starts with the shortest edge in E'_u. NMat finishes all the iterations when all non-deleted edges have been checked.

For example, in Fig 6, node p's neighbors are unevenly distributed in directions. Assuming the angle formed by p_1, p, and p_2 is larger than α, the longer edge (p, p_1) will be removed. As a result, the cost of finding q's nearest neighbor will be reduced to 4 search iterations and 10 distance calculations.

After the edge deletion process, the connectivity of I_G may be destroyed. To still guarantee the connectivity of I_G, NMat does the following: (1) NMat first detects the strongly connected components (SCCs) of I_G after the edge deletion process. (2) NMat finds out the node nearest to the geometric center of SCC in the embedding space E^d. NMat then spans a breadth-first-search (BFS) tree from the node. (3) For each leaf node of the BFS tree, NMat finds out its approximate nearest neighbors (ANNs) in other SCCs and add edges between leaf nodes and its ANNs.

4.3 Edge Judging

Recall that SMat includes a learning model EJud for judging whether an l-labeled edge exists between two given nodes u and v. An approach is to apply the embedding techniques for completing knowledge graphs [3, 29, 34]. Given the nodes u, v and label l, the embedding techniques train a learning model that computes their structural embeddings as S_u, S_v and S_l. We can obtain their relation as:

$$\mathrm{S}_l = \mathrm{S}_u \circ \mathrm{S}_v \tag{2}$$

where $|\mathrm{S}_l| = 1$ and $\circ$ denotes the Hadamard product.

By simply applying Equ. (2), we have two issues.

(1) S_l always has a probabilistic error due to the embedding techniques. Because we join multiple star matches to obtain the answer, multiple *judged edges* may be involved in this joining. Therefore, we could encounter a cascading error incurred by these edges after the joining.

(2) Nodes u and v contain rich information represented by their content vectors S_u and S_v. If we do not consider them as Equ. (2), the error can be further enlarged.

To solve the two issues, we design the learning model (Fig. 7) EJud that includes two components: symbolic enhancement and entanglement. The symbolic enhancement solves the issue of the cascading error. The entanglement fuses content and structural vectors into the model to return a more accurate judging.

Given two nodes u and v in G, EJud first applies any embedding method [3, 29, 34] to obtain initial structural vectors S_u and S_v. EJud then inputs S_u and S_v into the symbolic enhancement and entanglement as introduced as follows.

(1) Symbolic enhancement.

The idea of symbolic enhancement is to use more symbolic signals (i.e., existed labels in G) to enhance S_u and S_v.

Specifically, EJud first sets an one-hot vector $\mathbf{p}_u \in \{0,1\}^{1\times|V|}$ for node u and an adjacent matrix $\mathbf{M}_l \in \{0,1\}^{|V|\times|V|}$ for label l as their symbolic representations, where $\mathbf{M}_l^{(u,v)} = 1$ if $l(u,v) \in G$, otherwise $\mathbf{M}_l^{(u,v)} = 0$. EJud then yields a multi-hot vector with matrix multiplications (Equ. (3)) to represent the nodes linked with u via label l.

$$\mathbf{p}'_u = g(\mathbf{p}_u \mathbf{M}_l)^T \tag{3}$$

where $g(\cdot)$ is a normalization function: $g(\mathbf{x}) = \mathbf{x}/sum(\mathbf{x})$. We refer $\mathbf{p}'_u$ as u's symbolic vector, and each element of $\mathbf{p}'_u$ could be regarded as the probability of the corresponding node.

Further, assuming the node set with a non-zero probability in $\mathbf{p}'_u$ as SP_u, EJud employs an aggregation function (Equ. (4)) to compute a new vector $\mathrm{S}_u^{\mathbf{P}}$ as u's symbolic enhanced structural vector with an MLP function:

$$\mathrm{S}_u^{\mathbf{P}} = \sum_{i \in SP_u} \mathbf{p}_u^{\mathbf{i}'} \mathrm{MLP}(\mathrm{S}_i)\mathrm{S}_i \tag{4}$$

where $\mathbf{p}_u^{\mathbf{i}'}$ is the corresponding probability of node i in $\mathbf{p}'_u$. Therefore, for node u, the yielded new structural vector $\mathrm{S}_u^{\mathbf{P}}$ gathered more ground trues after the symbolic enhancement. Accordingly, EJud also obtains the new structural vector $\mathbf{S}_v^{\mathbf{P}}$ for node v.

(2) Entangling content and structural vectors.

The idea of entanglement is to fuse the content vector $\mathbf{C}_u$ (resp. $\mathbf{C}_v$) into $\mathbf{S}_u^{\mathbf{P}}$ (resp. $\mathbf{S}_v^{\mathbf{P}}$).

A simple way for entanglement is to concatenate $\mathbf{C}_u$ and $\mathbf{S}_u^{\mathbf{P}}$. Considering that the impacts on $\mathbf{S}_l$ from $\mathbf{C}_u$ and $\mathbf{C}_v$ to be distinct, EJud first introduces an MLP-based attention mechanism to calculate the relevance between $\mathbf{C}_u$ and $\mathbf{S}_l$ given by Equ. (5).

$$\begin{aligned}\text{Att}_{u,l} &= \text{Attention}(\mathbf{S}_l, \mathbf{C}_u) \\ &= \mathbf{V}^{\mathbf{T}}\tanh(\mathbf{W}\mathbf{S}_l + \mathbf{U}\mathbf{C}_u)\end{aligned} \tag{5}$$

where $\mathbf{V}^{\mathbf{T}}$, $\mathbf{W}$ and $\mathbf{U}$ are parameters to be trained, and $\tanh(\cdot)$ is an active function.

Accordingly, EJud can obtain the relevance $\text{Att}_{v,l}$ between $\mathbf{C}_v$ and $\mathbf{S}_l$. With the relevance, EJud entangles $\mathbf{C}_u$ and $\mathbf{S}_u^{\mathbf{P}}$ by concatenating $\mathbf{C}_u \cdot \text{Att}_{u,l}$ with $\mathbf{S}_u^{\mathbf{P}}$ given by Equ. (6).

$$\mathbf{E}_u = \text{Concat}(\text{Att}_{u,l} \cdot \mathbf{C}_u, \mathbf{S}_u^{\mathbf{P}}) \tag{6}$$

EJud can also obtain $\mathbf{E}_v$ for node v.

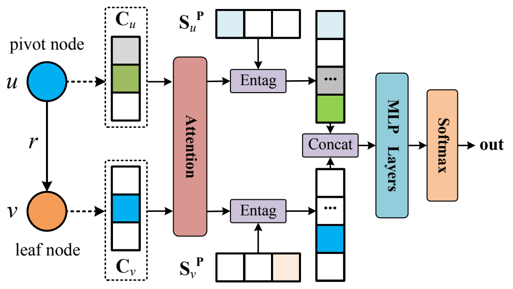


Figure 7: Process for building the model EJud.

(3) Model training.

With $\mathbf{E}_u$ and $\mathbf{E}_v$, EJud finally outputs a predicted label $\mathbf{l}_{\mathbf{out}}$ for edge (u, v) given by Equ. (7).

$$\mathbf{l}_{\mathbf{out}} = \text{softmax}(\text{MLP}(\text{Concat}(\mathbf{E}_u, \mathbf{E}_v))) \tag{7}$$

If $\mathbf{l}_{\mathbf{out}}$ is the same as the judged label l_e, EJud returns true and false otherwise.

The model could be trained by a simple cross entropy loss function, as given by Equ. (8):

$$Loss = -\mathbf{H}_l \log(\mathbf{l}_{\mathbf{out}}) \tag{8}$$

where $\mathbf{H}_l$ is the one hot vector of label l .

Fig. 7 shows the complete process for building the model EJud.

5 TOP-K JOIN & PATTERN DECOMPOSITION

A graph pattern Q can be decomposed to a set of star-shaped pattern SQ. Top-k answers to Q can be assembled by collecting the top matches of each star in SQ, followed by a multi-way join process.

5.1 Top-k Join

Given a graph pattern Q decomposed to a set of star patterns $SQ = \{sq_1, ..., sq_m\}$, the top-k answers to Q can be assembled by collecting the top matches of each sq_i, followed by a multi-way join process.

The procedure of star join (JoinK) first iteratively fetches k matches for each star and joins them with the existing matches for the other stars. JoinK then adds the join results to the priority queue R and keeps track of lower bound LB as the k-th match in R. If upper bound is smaller than the upper bound, JoinK removes sq_i from SQ. When SQ=∅, JoinK returns the first k results in R as the final answers. It can be seen that the efficiency of JoinK relies on the upper bound for each star introduced as follows.

Upper bound. The work [21] gives an upper bound for the joins of relational data. The upper bound is estimated as the sum of the scores of the last match in one list and the top-1 matches in all other lists. However, because the joint nodes exist in multiple stars, directly using this formula will cause joint nodes to be counted many times, thus making the calculated upper bound invalid. To tackle this issue, we propose an approach to compute a valid upper bound.

Let U be the set of the joint nodes for two stars q_1 and q_2, and A (resp. B) be the set of nodes that appear only in q_1 (resp. q_2). Then based on a parameter β, we introduce a ranking function scheme, denoted as $S'(h(q_1)) = S(h(A)) + \beta \cdot S(h(U))$ for q_1 and $S'(h(q_2)) = S(h(B)) + (1-\beta) \cdot S(h(U))$ for q_2, where $S(h(\cdot))$ is the similarity function given before.

Consider two matches lists L_1 and L_2. Regarding a list L_i of size n_i (i=1 or 2), denote h_{ij} as the jth ranked match in L_i. Then the upper bound can be defined as:

$$UB_1 = S'(h_{1n_1}) + S'(h_{21}), UB_2 = S'(h_{11}) + S'(h_{2n_2}) \tag{9}$$

When $\beta \in [0,1]$, one may verify that UB_1 and UB_2 are valid upper bounds for the search on q_1 and q_2, respectively. It is worth mentioning that the selection of β affects the number of matches to be searched for assembling. We determine its value in the experiments.

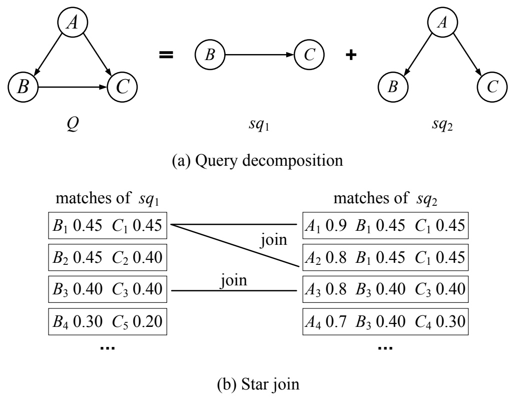


Figure 8: Pattern decomposition and star join.

Example 5.1. Assume that a pattern Q is decomposed into two stars sq_1 and sq_2 as shown in Fig.8(a). Their star matches are given in Fig.8(b). To identify the top-3 join matches, JoinK first fetches 3 matches for sq_1 and sq_2. Let parameter β=0.5. For sq_1, since nodes B and C are joint nodes, their match score is $B_1 = C_1 = 0.9 \times 0.5 = 0.45$. Because node A only appears in sq_2, match scores of node B and node C in sq_2 are multiplied by (1-0.5) (e.g. B_1=0.9×(1-0.5)=0.45). JoinK then obtains three join matches which are $A_1B_1C_1$, $A_2B_1C_1$ and $A_3B_3C_3$ with scores 2.7, 2.6 and 2.4, respectively. Due to $|R|$ =

3, JoinK updates lower bound as $LB = 2.4$. For sq_2, the upper bound is $A_3B_3C_3$+B_1C_1=1.6+0.9=2.5>2.4. Therefore, JoinK finds the next match $A_4B_3C_4$ for sq_2 and computes the upper bound as $A_4B_3C_4$+B_1C_1=1.4+0.9=2.3. Since 2.3<2.4, sq_2 can be removed from SQ. Based on the same principle, the upper bound of sq_1 is 2.3 and hence sq_1 can be removed from SQ. Finally the three results are $A_1B_1C_1$, $A_2B_1C_1$ and $A_3B_3C_3$, and the total depth is 6. □

5.2 Pattern Decomposition

Given a pattern Q decomposed to a set of star patterns $SQ = \{sq_1, ..., sq_m\}$, the total depth D=$\sum_{i=1}^{m} |L_i|$ for the m stars affects the performance of the matching and joining, where L_i is a list to maintain star matches of sq_i in a decreasing order. We expect to minimize the total depth D. In the section, we propose an algorithm DecQ for pattern decomposition that aims to minimize D.

Since all scores of star matches are generated online, it is very difficult to analyze the search depth accurately. We investigate several heuristics for DecQ as follows: (a) The number of decomposed star patterns should be as small as possible, which intuitively reduces the number of joins. (b) To make the upper bound estimation tighter in Equ. (9), we shall make $S(h_{in_i})$ as small as possible. In the following, we discuss the methods that we are used to achieve the two observations in DecQ.

Achieve observation 1. To achieve observation 1, we have to uncover the following problem: Let Q be a graph pattern, $SQ = \{sq_1, ..., sq_m\}$ be a set of stars such that any edge of Q belongs to one and only one $sq_i \in SQ$. We call SQ a *star cover* of Q. The problem is to find the minimum star cover of Q. It is not difficult to prove that finding the minimum star cover problem is polynomial equivalent to the minimum node cover problem which is NP-hard. As a result, our problem is an NP-hard problem.

DecQ can construct a star cover from a node cover in polynomial steps. There exists a 2-approximate algorithm [9] for the node cover problem. The algorithm works as follows. In each step, it randomly chooses an edge (u, v), adds u and v in the result, and removes all the edges incident to u or v. It repeats the process until all of the edges are removed. We can use the same process to create a 2-approximate star cover.

To realize observation 2, DecQ revises the algorithm by picking edges with higher *selectivity*.

Achieve observation 2. DecQ calculates a selectivity $f(a)$ of a node $a \in Q$ as follows: DecQ constructs an index I_a for each node a of G. Given a threshold γ, I_a maintains a set of nodes b such that $\delta(a, b) \leq \gamma$, i.e., $I_a = \{b|\delta(a, b) \leq \gamma, b \in G\}$. I_a is compressed and stored in a hash table. DecQ sorts I_a in a decreasing order of $\delta(a, b)$, and then obtains w decreasing similarities $\delta_1, ..., \delta_w$. DecQ then defines $f(a) = w/(\delta_w - \delta_1)$. Intuitively, $(\delta_w - \delta_1)/w$ measure the average score decrement. Therefore, the larger $f(a)$ is, the larger the score decrement is. To this end, in each step, DecQ only picks an edge (u, v) with the largest $f(u) + f(v)$.

For example, based on the two observations, the pattern Q in Fig.8(a) can be decomposed into 2 stars with the smallest number of stars and tighter upper bound.

6 EVALUATION

Using real-life graphs, we evaluated the accuracy, efficiency and scalability of our algorithms.

6.1 Setup

Test-bed. All codes are written in C++, and are compiled by g++ 6.5. All experiments are conducted on a Linux server with a 4-core Intel Core i7-880 3.06GHz CPU, 256 GB of memory, and 1TB of HDD. The training model is implemented with Pytorch framework on RTX3090 GPU.

Real-life graphs. We used three real-life multi-modal knowledge graphs: (a) IMGpedia [11] is a large-scale linked data set with a large number of visual information from images in the Wikimedia Commons dataset. It contains 502 million nodes, 3,119 million edges and 14.8 million visual contents. (b) Richpedia [31] is a multi-modal knowledge graph by distributing sufficient and diverse images to textual entities in Wikidata. It contains 3.1 million nodes, 119.7 million edges and 2.9 million images. (c) YAGO15K [23] is a small graph created from the YAGO dataset, and contains numeric literals and images as attributes. It contains 15.2 million nodes, 112.9 million edges and 11.2 million images. All the images in these graphs are transformed into 128-dimensional vectors by the state-of-the-art deep leaning model.

Pattern workload. We first generate star patterns, and then extend the stars by adding nodes and edges to generate patterns with complex structure, e.g., cliques, circles and multiple stars. We mine frequent labels and images from the real-life graphs and randomly assign them to the patterns. We use Qi to denote the size of a pattern Q, where i is the number of nodes of Q. In the experiments, we set the pattern sets as $Q2$, $Q4$, $Q6$, $Q8$ and $Q10$, and $Q6$ is the default one. The average numbers of images assigned to the five pattern sets are 1, 2, 2, 3 and 4. We also set the values of top-k as 1, 10, 50, 100 and 150, and top-50 is the default one. We report the average results of all indicators by performing five repeated trials.

Measurements. We use *recall rate*, which equals the ratio of the number of successfully returned top-k matches to k, to measure the search accuracy. We also use the running time to measure the search performance, i.e., the total time of a pattern to be issued until all answers to be returned.

Algorithms. We compare our proposed approach NSMatch with the following algorithms.

(1) NSnop is just NSMatch *without* the optimized techniques, i.e., edge deletion and edge judging model.

(2) SubISO is a state-of-the-art subgraph isomorphism algorithm [16]. To solve our problem, SubISO first determines all matches of Q in G. SubISO then ranks the matches by computing $S(h(Q))$ and returns the top-k matches.

(3) GraphTA advocates the threshold algorithm [10] to identify top-k matches from a graph. GraphTA first initializes a candidate list for each pattern node and sorts each list following the similarity function. From the head of each sorted list, GraphTA then iteratively starts an exploration based subgraph isomorphism search to expand the node match until k matches are identified.

Parameter setting. The search length f and the size l of the candidate pool for the ANNS are sets to 40 and 500, respectively. The angle α in the strategy of edge deletion is set to 60. To train the EJud model, we set the embedding dimension of node and edge to be 1024, respectively; the hidden state dimension of MLP is set to

1024; the learning rate is set to be 10^{-5}; and the training batch size is 64, while the negative sample size is 128. The β in the joining algorithm and the search depth h of ANNS will be determined in the experiments.

6.2 Experimental Results

Exp-1: determine parameters β and h. To determine β and h, we vary their values and report the running time of NSMatch on YAGO15K, Richpedia and IMGpedia, respectively.

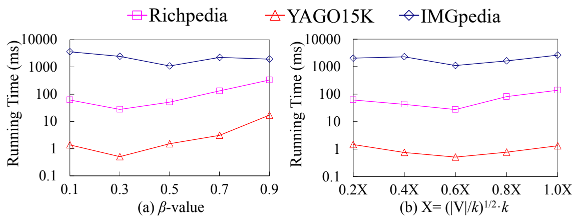


Figure 9: Running time of NSMatch w.r.t. β and h.

Varying β. β is an important parameter, in the joining algorithm, whose value influences the entire performance. Fig. 9(a) depicts the running time of NSMatch by varying β from 0.1 to 0.9. It shows that a well selected β value indeed leads to less runtime. Considering each dataset, the best performance can be achieved when $\beta = 0.3$ for Richpedia, $\beta = 0.3$ for YAGO15K and $\beta = 0.5$ for IMGpedia, respectively. We thus use these values for NSMatch in the experiments.

Varying h. The value of h affects the star matching. We set h to $0.2X$–$1.0X$, where $X = \sqrt{(|V|/k)} \cdot k$. Fig. 9(b) shows the running time of NSMatch on the three graphs. The result tells us the following. NSMatch achieves the best performance at $h = 0.6X$ for all three graphs. h determines the trade-off among the neural and symbolic calculations, which is also reflected in the figure. A lager h leads to more neural computations (i.e., embedding search) and thus fewer symbolic calculations (i.e. graph traversal) and vice versa. We therefore set $h = 0.6X$ in the experiments.

Exp-2: varying $|Q|$. Varying $|Q|$ (pattern size) from 2 to 10, we report the running time and recall rates of different algorithms.

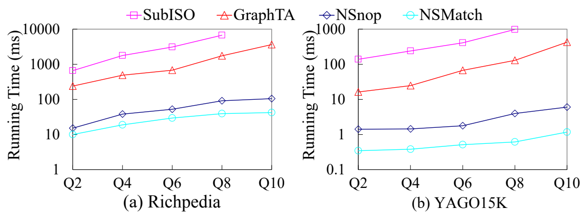


Figure 10: Runtime w.r.t. different pattern sizes.

Running time. Fig. 10 shows the results over both Richpedia and YAGO15K. As shown in the figure, (1) as the pattern becomes larger, the running time of SubISO and GraphTA grows in exponential, while NSnop and NSMatch are less sensitive; (2) NSMatch improves GraphTA and SubISO better over larger patterns, and is 22 times and 63 times faster than GraphTA and SubISO for even single edge pattern with 2 nodes; and (3) NSMatch is 2.5 times faster than NSnop on average. The reason is that NSnop does not optimize the ANN search as NSMatch and the ANN search influences the entire performance greatly.

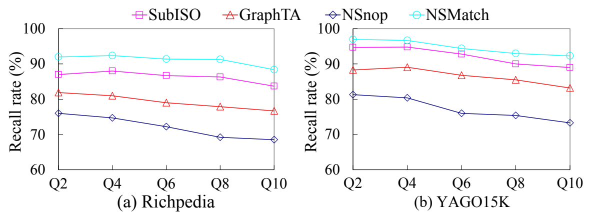


Figure 11: Recall rates w.r.t. different pattern sizes.

Recall rates. Fig. 11 plots the recall rate curves of the four algorithms on Richpedia and YAGO15K. As shown in the figure, (1) as the pattern becomes larger, the recall rates of the four algorithms decline slowly; and (2) NSMatch has the highest recalls (above 92% on average), and NSnop has the lowest recalls (below 80% on average). This is because NSMatch includes the edge judging model to complete missed edges during the star matching, while other three algorithms neglect this. NSnop also applies the approximate NN search and thus obtains the lowest recall.

Exp-3: varying top-k. Varying k from 1 to 150, we report the results of recall rates and running time of different algorithms.

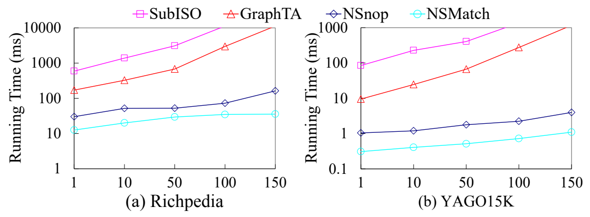


Figure 12: Running time w.r.t. different top-k values.

Running time. Fig. 12 shows the results over both Richpedia and YAGO15K. From the figure, we can see that the running time of GraphTA and SubISO grows dramatically when k increases. Indeed, both GraphTA and SubISO use top scored node matches to find complete matches, which incurs considerable useless enumeration and traversal, especially for larger k. The top scored node matches might not lead to the best matches of the pattern. In contrast, NSMatch and NSnop outperform all other methods in orders of magnitude, and their performance is much less sensitive to the growth of k. We observe that the main bottleneck for NSnop is the expensive ANNS, especially for larger k and denser graphs (Richpedia). NSMatch copes with this quite well: almost all results are acquired in 0.1 second.

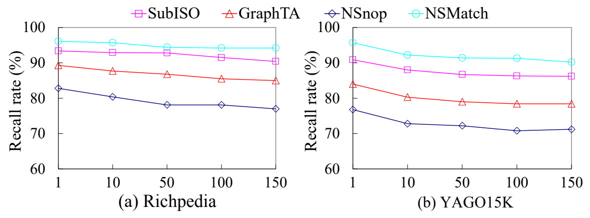


Figure 13: Recall rates w.r.t. different top-k values.

Recall rates. Fig. 13 plots the recall rate curves of the four algorithms on Richpedia and YAGO15K. As shown in the figure, (1) as k becomes larger, the recall rates of the four algorithms decrease but not obviously; and (2) NSMatch outperforms the other methods consistently, which shows the robustness of NSMatch in solving the problem of approximate search over incomplete graphs.

Exp-4: varying $|G|$. This experiment studies the scalability of the algorithms over IMGpedia. Specifically, we generate graphs from IMGpedia by varying $|N|$ (number of nodes) from 10M to 500M and $|E|$ (number of edges) from 100M to 3000M. Under these graphs, we report the results of running time and recall rates of different algorithms.

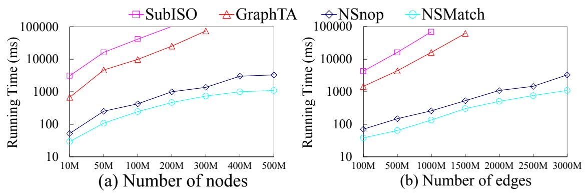


Figure 14: Running time w.r.t. different graph sizes.

Running time. Fig. 14 reports the results, from which we find that (1) when the graph size increases, the running time of all the algorithms increases, more obvious for edges than for nodes; (2) NSMatch and NSnop outperform their competitors by at least two orders of magnitude; and (3) NSMatch further improves NSnop by 75%–85%, as expected.

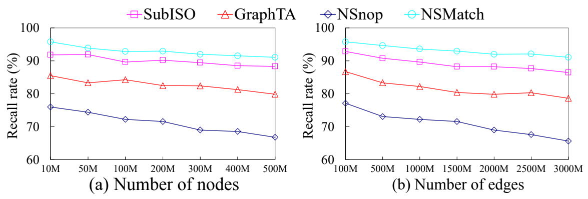


Figure 15: Recall rates w.r.t. different graph sizes.

Recall rates. Fig. 15 gives the recall rates with different graph sizes. As shown in the figure, (1) with the increase of graph size, all the plots decrease but NSMatch is the slowest; and (2) NSMatch has a high recall (i.e., 91%) even for a graph of 3000M edges, and has higher recalls 10% 15% and 20% than other three algorithms on average. These experimental results show that our proposed method is very scalable with respect to different graph sizes.

7 RELATED WORK

We categorize the related work as follows.

Subgraph search. There have been a large number of algorithms developed for subgraph search so far, which can generally be classified into symbolic methods and neural approaches. In the first category, the symbolic methods include the classical backtrack search [5, 16, 17, 30, 40], especially determining the optimal search order by the dynamic programming; and the encoding and indexing techniques [2, 19, 28, 36, 42, 45] for which the nodes within a distance of each node are encoded as signatures and indices. In the second category, many neural methods (e.g., TransE [3], ConE [44] and BetaE [27] convert nodes and edges into structural embeddings that can be used to solve the incompleteness for unimodal graphs. With respect to link prediction for multimodal graphs, the work [37] extends TransE [3] to obtain visual representations that correspond to the graph entities and structural information separately. The series models [25, 33, 46] further propose several fusion strategy to encode the visual and structural features into a unified embedding space.

Approximate nearest neighbors search (ANNS). In this paper, we focus on graph-based methods since extensive experimental studies have shown their superiority on search performance among the ANNS methods [13, 24, 32].

Among graph-based methods, two state-of-the-art works are HNSW [24] and NSG [13]. Hierarchical navigating small world (HNSW) [24] uses a hierarchical structure built on a navigating small world graph. HNSW conducts the search in the direction from the top layer to the base layer and ensures that a locally nearest graph node can be found in each layer. Navigating spreading-out (NSG) [13] uses a single-layer graph structure. Specifically, the authors in [13] proposed monotonic relative navigating graph (MRNG) based on ideas of MSNET [1] and RNG [6]. Recently, inspired by the superior performance of HNSW, other graph-based methods include theoretical analysis of KNN graph[26], a KNN graph built on Jaccards index, and multi-core capacity-optimized multi-store ANN algorithm [43].

NSMatch differs from all the prior works in the following. (1) Traditional subgraph matching is based on the pure symbolic semantics (i.e., subgraph isomorphism), whereas NSMatch is defined from the neural-symbolic semantics (i.e., top-k subgraph matching based on the neural embeddings). (2) Algorithms for traditional subgraph matching cannot be used to process NSMatch. Despite the increased expressive power of NSMatch, its complexity is no harder than the traditional subgraph matching. We provide practical algorithms to support a good scalability of NSMatch over large graphs. (3) It may lower the search efficiency to directly apply the existing ANNS algorithms over NSGDs, because they do not consider the skewed distribution of embeddings of NSGDs. (4) Our approach enables the neural and symbolic reasoning to enhance each other to alleviate the cascading error and to include content vectors which do not be paid attention to by the existing works on graph incompleteness.

8 CONCLUSION

We have studied neural-symbolic subgraph matching (NSMatch). The novelty of this work consists of the following: (1) neural-symbolic graph database (NSGD) model to support applications of multi-modal knowledge graphs and multi-modal social medias; (2) a general and efficient algorithmic framework to process the NSMatch; (3) strategies of edge deletion to speed up the graph-based ANNS; and (4) a neural-symbolic learning model to complete the missing edges of the NSGD. Our experimental study has verified that the method is promising in practice.

One topic for future work is to study the incremental NSMatch. Another topic is to investigate more pattern types (e.g., shortest path and clique finding) over large NSGDs.

ACKNOWLEDGMENTS

Ye Yuan is supported by the National Key R&D Program of China (Grant No.2022YFB2702100), the NSFC (Grant Nos. 61932004, 62225203, U21A20516) and the DITDP (Grant No. JCKY2021211B017). Jianbin Qin is supported by the National Key R&D Program of China (Grant No. 2021YFB3301502 and 2021YFB3301503).

REFERENCES

[1] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, volume 93, pages 271–280, 1993.

[2] B. Bhattarai, H. Liu, and H. H. Huang. Ceci: Compact embedding cluster index for scalable subgraph matching. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1447–1462, 2019.

[3] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

[4] X. Chen, N. Zhang, L. Li, S. Deng, C. Tan, C. Xu, F. Huang, L. Si, and H. Chen. Hybrid transformer with multi-level fusion for multimodal knowledge graph completion. *arXiv preprint arXiv:2205.02357*, 2022.

[5] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(10):1367–1372, 2004.

[6] D. Dearholt, N. Gonzales, and G. Kurup. Monotonic search networks for computer vision databases. In *Twenty-Second Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 548–553. IEEE, 1988.

[7] L. Dietz, A. Kotov, and E. Meij. Utilizing knowledge graphs for text-centric information retrieval. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pages 1387–1390, 2018.

[8] W. Dong, M. Charikar, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In S. Srinivasan, K. Ramamritham, A. Kumar, M. P. Ravindra, E. Bertino, and R. Kumar, editors, *Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011*, pages 577–586. ACM, 2011.

[9] D. H. (ed.). *Approximation algorithms for NP-Hard problems.* PWS, 1997.

[10] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[11] S. Ferrada, B. Bustos, and A. Hogan. Imgpedia: a linked dataset with content-based analysis of wikimedia images. In *International Semantic Web Conference*, pages 84–93. Springer, 2017.

[12] C. Fu, C. Wang, and D. Cai. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[13] C. Fu, C. Xiang, C. Wang, and D. Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143*, 2017.

[14] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness.* W.H.Freeman, 1979.

[15] W. Gong, E.-P. Lim, and F. Zhu. Characterizing silent users in social media communities. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 9, pages 140–149, 2015.

[16] M. Han, H. Kim, G. Gu, K. Park, and W.-S. Han. Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1429–1446, 2019.

[17] W.-S. Han, J. Lee, and J.-H. Lee. Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 337–348, 2013.

[18] S. Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1-3):335–346, 1990.

[19] H. He and A. K. Singh. Graphs-at-a-time: query language and access methods for graph databases. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 405–418, 2008.

[20] J. Huang, W. X. Zhao, H. Dou, J.-R. Wen, and E. Y. Chang. Improving sequential recommendation with knowledge-enhanced memory networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 505–514, 2018.

[21] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top-k join queries in relational databases. *VLDB J.*, 13(3):207–221, 2004.

[22] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[23] Y. Liu, H. Li, A. Garcia-Duran, M. Niepert, D. Onoro-Rubio, and D. S. Rosenblum. Mmkg: multi-modal knowledge graphs. In *European Semantic Web Conference*, pages 459–474. Springer, 2019.

[24] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

[25] H. Mousselly-Sergieh, T. Botschen, I. Gurevych, and S. Roth. A multimodal translation-based approach for knowledge graph representation learning. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, pages 225–234, 2018.

[26] L. Prokhorenkova and A. Shekhovtsov. Graph-based nearest neighbor search: From practice to theory. In *International Conference on Machine Learning*, pages 7803–7813. PMLR, 2020.

[27] H. Ren and J. Leskovec. Beta embeddings for multi-hop logical reasoning in knowledge graphs. *Advances in Neural Information Processing Systems*, 33:19716–19726, 2020.

[28] H. Shang, Y. Zhang, X. Lin, and J. X. Yu. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *Proceedings of the VLDB Endowment*, 1(1):364–375, 2008.

[29] Z. Sun, Z. Deng, J. Nie, and J. Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[30] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, 1976.

[31] M. Wang, H. Wang, G. Qi, and Q. Zheng. Richpedia: a large-scale, comprehensive multi-modal knowledge graph. *Big Data Research*, 22:100159, 2020.

[32] M. Wang, X. Xu, Q. Yue, and Y. Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proc. VLDB Endow.*, 14(11):1964–1978, 2021.

[33] Z. Wang, L. Li, Q. Li, and D. Zeng. Multimodal data enhanced representation learning for knowledge graphs. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[34] Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[35] Y. Wei, X. Wang, L. Nie, X. He, R. Hong, and T.-S. Chua. Mmgcn: Multi-modal graph convolution network for personalized recommendation of micro-video. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 1437–1445, 2019.

[36] Y. Wu, S. Yang, and X. Yan. Ontology-based subgraph querying. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 697–708. IEEE, 2013.

[37] R. Xie, Z. Liu, H. Luan, and M. Sun. Image-embodied knowledge representation learning. *arXiv preprint arXiv:1609.07028*, 2016.

[38] Z. Yang. Biomedical information retrieval incorporating knowledge graph for explainable precision medicine. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2486–2486, 2020.

[39] Y. Yuan, L. Chen, and G. Wang. Efficiently answering probability threshold-based shortest path queries over uncertain graphs. In *Database Systems for Advanced Applications: 15th International Conference, DASFAA 2010, Tsukuba, Japan, April 1-4, 2010, Proceedings, Part I 15*, pages 155–170. Springer, 2010.

[40] Y. Yuan, G. Wang, L. Chen, and H. Wang. Efficient subgraph similarity search on large probabilistic graph databases. *Proceedings of the VLDB Endowment*, 5(9), 2012.

[41] Y. Yuan, G. Wang, L. Chen, and H. Wang. Efficient keyword search on uncertain graph data. *IEEE Transactions on Knowledge and Data Engineering*, 25(12):2767–2779, 2013.

[42] Y. Yuan, G. Wang, H. Wang, and L. Chen. Efficient subgraph search over large uncertain graphs. *Proceedings of the VLDB Endowment*, 4(11):876–886, 2011.

[43] M. Zhang and Y. He. Grip: Multi-store capacity-optimized high-performance nearest neighbor search for vector search engine. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1673–1682, 2019.

[44] Z. Zhang, J. Wang, J. Chen, S. Ji, and F. Wu. Cone: Cone embeddings for multi-hop reasoning over knowledge graphs. *Advances in Neural Information Processing Systems*, 34:19172–19183, 2021.

[45] P. Zhao and J. Han. On graph query optimization in large networks. *Proceedings of the VLDB Endowment*, 3(1-2):340–351, 2010.

[46] Y. Zhao, X. Cai, Y. Wu, H. Zhang, Y. Zhang, G. Zhao, and N. Jiang. Mose: Modality split and ensemble for multimodal knowledge graph completion. *arXiv preprint arXiv:2210.08821*, 2022.